```python
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 23 15:34:41 2019

@author: Alexm
"""

import numpy as np
import matplotlib.pyplot as plt


print ("Alexandra Mulholland 17336557")

#Aim is to fit a data set for measurements of radioactive decay

#time (in seconds) = [5,15,25,35,45,55,65,75,85,95,105,115]= y axis
#number of decays = [32,17,21,7,8,6,5,3,4,1,5,1]= x axis

#PART 1
#Plot the data set, i.e. time vs. number of decays, using a linear scale on
#on both axes

time = np.array([5,15,25,35,45,55,65,75,85,95,105,115])
Number_of_decays = np.array([32,17,21,7,8,6,5,3,4,1,5,1])



plt.figure(1)
plt.plot(time, Number_of_decays, 'g-o')
plt.ylabel('Number of decays')
plt.xlabel('Time /seconds')
plt.title('Plot of number of decays versus time')
plt.grid(True)
plt.show()

#PART 2

#Plot log-linear-- log y axis and linear x axis = straight line
#why? what equation?
#estimate offset and slope of line

plt.figure(2)
plt.plot(time, np.log(Number_of_decays),'b-o')
plt.ylabel('ln(Number of decays)')
plt.xlabel('Time /seconds')
plt.title('Plot of natural log of the number of decays versus time')
plt.grid(True)
plt.show()

#The plot appears to be roughly a straight line, showing the number of decays
#decreases with time
#Because of the small sample size, plot has many anomalies
#therefore, I will plot line of best fit (not necessary but will use to compare later)
#y = mx + c
#want to find m and c
plt.figure(3)
plt.plot(np.unique(time), np.poly1d(np.polyfit(time, np.log(Number_of_decays), 1))(np.unique(time)),'pink')
plt.plot(time, np.log(Number_of_decays),'b-o')
plt.ylabel('ln(Number of decays)')
plt.xlabel('Time /seconds')
plt.title('Natural log of number of decays vs time with line of best fit')
plt.grid(True)
plt.show()
#By eye the intercept seems to be 3.5 (i.e. ln(32))
#the gradient, from the line of best fit, is rise/run=(3-0.7)/(10-100)=-0.0256
#Relevant equation N(t)=N(0)e^(-lambda x t)
#plotting ln(N(t)) versus t
#rearranging and solving for t gives y=mx+c----lnN(t)=lnN(0)-(lambda x t)


#PART 3

```

```python
#carrying out a linear regression/least square fit of the data
#Defining each term used, to find paramters a and b
#which in this case are ln(N(0)) and -lambda respectively
A = np.sum(time)
B = np.sum(np.square(time))
C = np.sum(np.log(Number_of_decays))
D= np.sum(np.multiply(np.log(Number_of_decays),time))
#defining number of data points N
N=12

#Solving the two equations by rearranging
#See working out

a= (D*A-(B*C))/((A*A)-(N*B))
b= (C-(a*N))/(A)

print('Paramater a:',a)
print('Parameter b:',b) #b1
print('estimated gradient=',(3-0.7)/(10-100))


#have our fit parameters, now must define the line
#which will be of the form y=mx+c ie f=a+bt where f is the decay
def function(time):
    f = b*time + a
    return f


#PART 4

#Plotting this linear regression
plt.figure(4)
plt.plot(time,function(time),'r', label='linear regression fit')
plt.plot(time,np.log(Number_of_decays),'b-o', label='ln(number of decays) vs time')
plt.xlabel('Time /seconds')
plt.ylabel('ln(Number of decays)')
plt.title('Natural log of number of decays vs time with linear regression line')
plt.grid(True)
plt.legend()
plt.show()



#We see that the intercept is close to predicted- value of 3.34089 (a)
#the gradient, which is b which equals -lambda, has a value of -0.03 roughly
#Which is fairly close to my expected value of -0.0256
#Value of the decay constant, with b=-lambda, is 0.0267033 s^-1

#PART 5
#Use the values for your two fit parameters to plot the corresponding
#decay function in your initial (linear scale) plot of the number of decays vs time.
#using N(t)=N(0)e^-(lambda x t)
#where b=-lambda and a=ln(N(0))=3.340898909967487
#therefore, will define a third parameter c in order to plot linearly
#and define it so it is just N(0)
#c=e^3.3408...
print(np.exp(3.340898909967487))

c = 28.244504564174118
def function2(time):
    g= c*np.exp(b*time)
    return g

plt.figure(5)
plt.plot(time,function2(time),'orange',label= 'Radioactive decay function')
plt.plot(time,Number_of_decays,'g-o',label='Number of decays versus time data')
plt.title('Plot of original data along with the decay function using fit parameters')
plt.xlabel('Time /seconds')
plt.ylabel('Number of decays')
plt.legend()
plt.grid(True)
plt.show()
```

```
#PART 6

#performing a non-linear least square fit of the data
import scipy.optimize as optimization

# Initial guess of fit parameters
c=0
b=0
guess=[c,b]
# define your nonlinear function (containing two fit parameter)
#Using c as this was the parameter used in the non linear fit
def function3(time,c,b):
    h = c*np.exp(b*time)
    return h

#Performing the fitting
fitparameter=optimization.curve_fit(function3,time, Number_of_decays, guess)[0]

#accessing the fit parameters
c=fitparameter[0]
b=fitparameter[1]

print('c=',fitparameter[0])
print('b=',fitparameter[1]) #b2

plt.figure(6)
plt.plot(time,Number_of_decays,'g-o',label='Radioactive decay data plot')
plt.plot(time,function3(time,c,b),'yellow',label='non-linear least square fit')
plt.title('Non-linear least square fit of data, along with original data')
plt.xlabel('Time /seconds')
plt.ylabel('Number of decays')
plt.grid(True)
plt.legend()
plt.show()

print('average decay constant value from 2 methods: (b1+b2)/2 is',(-0.026703363333904266+fitparameter[1])/2)

#c=35.365... meaning a=3.565740546
#b=-lambda=-0.032529...
#describes the data set effectively, similar parameter values
```