Erasmus Mundus JMD Nuclear Physics

Computational and Numerical Physics Exercises

Craig Michie

First of all, Import all Packages for use and Future Use

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import scipy
import math
import random
np.warnings.filterwarnings('ignore')
from mpl_toolkits import mplot3d
```

Exercise 1 ¶

In [2]:
```
'''
This portion of code reads the file of the weather for future use and will show the first 5 rows.
'''
df = pd.read_csv('T_Agrinio_EM.csv',header = 0, index_col = 0)
df.head()

#to ensure all the data points were read
#df.head(136)
```

Out[2]:

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12
year												
1961	8.56	9.18	10.86	14.62	18.13	21.98	25.97	25.33	21.75	17.61	13.15	10.70
1962	8.48	8.94	10.21	13.93	17.34	22.34	26.51	26.01	21.45	17.65	13.10	9.33
1963	9.30	9.85	10.75	13.45	17.71	22.24	25.52	25.44	22.35	16.57	13.41	8.56
1964	8.67	8.42	10.49	14.27	17.52	21.87	25.25	25.19	22.32	17.99	13.52	9.03
1965	7.59	8.05	10.68	13.45	17.64	22.75	25.90	25.53	22.26	16.99	13.31	10.42

Exercise 1.1: Determine the Minimum and Maximum Temperature for the Month of March for this Given Set

In [3]:
```
'''
First, to create a list from the table above of all the values in march, which is given by t3.
By using Pandas the values in the table are z values which would be organized in a 1x136 array,
So will have to append those values to a list to easily acquire the maximum value.
'''
z = df[['t3']][:136].values
List = []
for n in range(136):
    List.append(float((z[n])))
    n = n+1

print('The maximum March temperature from the dataset is:',np.max(List),'\n' 'The minimum March te
mperature from the dataset is: ',np.min(List))
print('The average March Temperature from the dataset is:','%.2f'%np.average(List))
```

```
The maximum March temperature from the dataset is: 14.44
The minimum March temperature from the dataset is:  9.95
The average March Temperature from the dataset is: 11.90
```

Exercise 1.2: Check the info on the numpy mean function and compute the average monthly temperatures and the average annual temperatures for the given dataset (Hint: axis option)

In [4]:
```
'''
To save Scrolling, I have # the line of code to show the info on the mean function
to check it. simply un-# it.
'''
#np.info(np.mean)
```

Out[4]:
```
'\nTo save Scrolling, I have # the line of code to show the info on the mean function\nto check i
t. simply un-# it.\n'
```

In [47]:
```
'''
This calculates the average temperature of each year by taking the array of all values,
which are seperated by each year and calculates the mean of the column which is equal to
the average temperature over that year...
'''
np.array(df)
z = np.array(df)
AnnualAverage = []
for n in range(136):
    l = '%.2f' %(np.average(z[n]))
    AnnualAverage.append(float(l))
    n = n+1


print(AnnualAverage)

'''
Expanding on the above code by transposing the array we acquire an array with lists of the monthly
average temperatures

'''

TransposedZ = z.T #this transposes the array
MonthlyAverage = []

for n in range(12):
    l = '%.2f'%(np.mean(TransposedZ[n]))
    MonthlyAverage.append(l)
    n= n+1

'''
We see that the average March temperature is the same as 1.1 so we know these values are correct.
'''
print(MonthlyAverage)
print('The average March Temperature from the dataset is:',MonthlyAverage[2])
```

```
[16.49, 16.27, 16.26, 16.21, 16.21, 15.95, 16.53, 16.25, 16.2, 16.24, 15.7, 15.96, 16.39, 16.09,
16.25, 16.34, 16.03, 16.41, 16.25, 16.5, 16.23, 15.71, 16.19, 16.01, 16.19, 16.25, 16.37, 16.18,
16.44, 16.53, 16.52, 16.93, 17.12, 16.36, 16.8, 16.64, 16.69, 16.99, 16.79, 16.83, 16.7, 17.02, 1
6.8, 16.91, 16.81, 16.52, 16.79, 16.88, 17.36, 17.01, 16.76, 16.93, 17.26, 16.84, 16.63, 17.39, 1
7.3, 17.06, 17.34, 16.75, 16.77, 17.18, 16.98, 17.29, 17.42, 17.0, 17.18, 17.31, 17.3, 17.52, 17.
77, 17.87, 17.91, 17.78, 18.02, 17.71, 18.06, 17.62, 17.84, 17.93, 17.81, 17.64, 18.33, 18.04, 1
7.72, 18.48, 18.18, 18.43, 18.51, 18.49, 18.09, 18.55, 18.71, 18.55, 18.74, 18.83, 18.71, 18.43,
18.55, 18.83, 18.52, 18.71, 18.77, 18.71, 18.85, 19.3, 18.88, 18.98, 18.87, 19.02, 18.94, 19.19,
18.88, 18.77, 19.12, 19.33, 19.26, 19.42, 19.47, 19.94, 19.64, 19.62, 19.45, 19.44, 19.27, 19.68,
19.36, 19.94, 20.02, 20.09, 19.95, 19.58, 19.72, 19.99, 19.97, 19.89]
['9.58', '9.94', '11.90', '14.89', '19.22', '24.04', '27.64', '27.63', '23.93', '18.84', '14.18',
'10.70']
The average March Temperature from the dataset is: 11.90
```

Exercise 1.3: Get the maximum and minimum monthly temperatures for year 2000 and at what months did these temperatures occur

In [6]:
```
'''
By looking at the list, it's easy to see that the year 2000 will occur at line 39,
... craig remember to create a loop to locate this line instead though
'''
Y2000List = []
p = df.head(136)[39:40].values
print(p)
print('The lowest Temperature for the year 2000 is:',np.min(p))
print('The maximum Temperature for the year 2000 is:', np.max(p))
```

```
[[ 9.18 10.56 11.81 14.32 17.55 22.93 25.56 25.73 21.72 17.47 14.54 10.54]]
The lowest Temperature for the year 2000 is: 9.18
The maximum Temperature for the year 2000 is: 25.73
```

Exercise 2

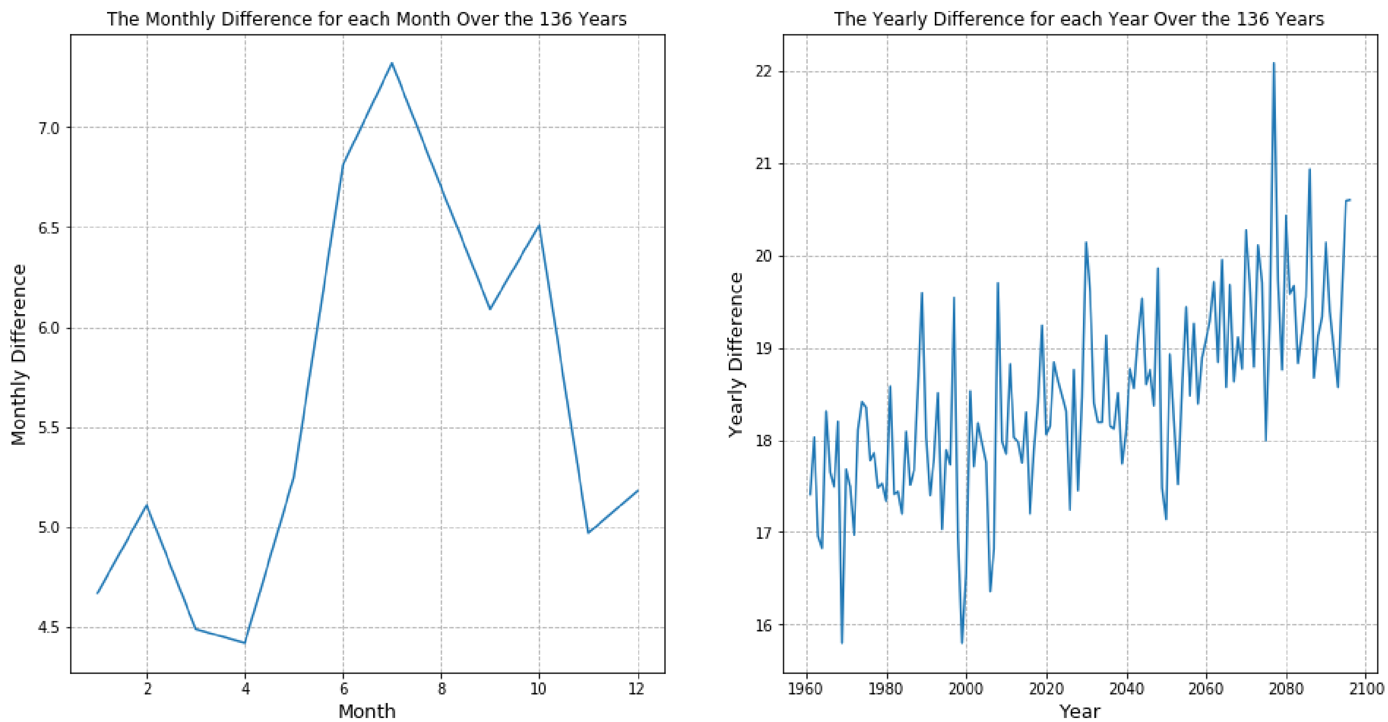
Exercise 2.1: Plot the monthly and annual difference between max and min temperatures as a function of the month (1-12) and the year (1961-2096), respectively

In [7]:

```
'''
We first have to evaluate the maximum and minumum temperatures of each month and each year.
This expands on the work done in exersise 1.3 which evaluated the maximum and minimum of a single
 year,
we'll want to create a loop that will determine the maximum and minimum, take the difference and a
dd
that difference to a list. For each month and each year
'''

'''
This block of code sets up the empty lists and bounds for the graphs
'''
x = []
f = []
for n in range(1,13):
    x.append(n)
    n=n+1


for n in range(1961,2097):
    f.append(n)
    n=n+1

Y_diff = []
M_diff = []

TransposedZ = z.T


for n in range(136):
    '''
    This loop determines the difference between the lowest and highest temperature of each
    year and appends them to a list to be used for the annual difference as a function of the year
    '''
    minn = np.min(z[n])
    maxx = np.max(z[n])

    Diff = maxx - minn
    Y_diff.append(float('%.2f'%Diff))



for n in range(12):
    '''
    This loop determines the difference between the lowest and highest temperature of each month
    over the 136 years and appends them to a list to be used in the monthly differences as a funct
ion
    of the month
    '''
    mminn = np.min(TransposedZ[n])
    mmaxx = np.max(TransposedZ[n])

    Dif = mmaxx - mminn
    M_diff.append(float('%.2f'%Dif))



'''
Plotting Code, first chunk is the monthly average and second chunk is yearly average
'''

fig, axarr = plt.subplots(nrows = 1, ncols = 2, figsize = (16,8))

axarr[0].plot(x,M_diff)
axarr[0].grid(linestyle = 'dashed')
axarr[0].set_xlabel('Month', fontsize=13)
axarr[0].set_ylabel('Monthly Difference', fontsize= 13)
axarr[0].set_title('The Monthly Difference for each Month Over the 136 Years')



#axarr[1].set_yticks(range(11),['10','11','12','13','14','15','16','17','18','19','20'])
```

```python
#axarr[1].set_ylim(10,30)
axarr[1].grid(linestyle='dashed')
axarr[1].set_xlabel('Year', fontsize=13)
axarr[1].set_ylabel('Yearly Difference', fontsize= 13)
axarr[1].set_title('The Yearly Difference for each Year Over the 136 Years')
axarr[1].plot(f,Y_diff)
```

Out[7]: [<matplotlib.lines.Line2D at 0x11ebe2ac8>]



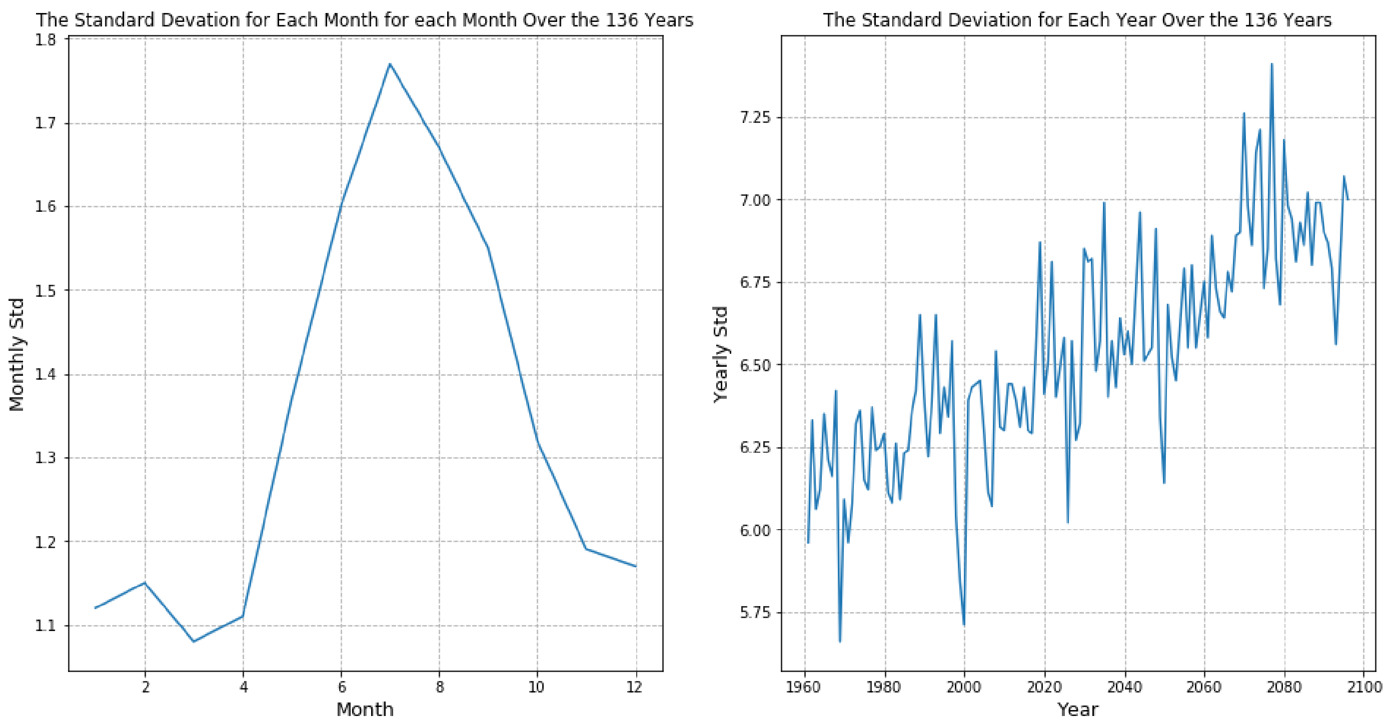
Exercise 2.2: Plot the standard deviation of the monthly and annual temperatures as a function of the month (1-12) and the year (1961-2096), respectively

In [8]:
```python
'''
We know numpy has a function for standard deviation so we jut have to integrate it into a loop and
append it to a list for graphing
'''
StdForYears = []

for n in range(136):
    '''
    This loop evaluates the standard deviation for each year and appends it to a list
    '''
    StdForYears.append(float('%.2f'%np.std(z[n])))
    n=n+1
print(StdForYears[:12])

StdForMonths = []
for n in range(0,12):
    '''
    This list evaluates the standard deviation for each month and appends it to a list
    '''
    StdForMonths.append(float('%.2f'%np.std(TransposedZ[n])))
    n=n+1

print(StdForMonths[:12])
```

```
[5.96, 6.33, 6.06, 6.12, 6.35, 6.21, 6.16, 6.42, 5.66, 6.09, 5.96, 6.08]
[1.12, 1.15, 1.08, 1.11, 1.37, 1.6, 1.77, 1.67, 1.55, 1.32, 1.19, 1.17]
```

```
In [9]: fig, axarr = plt.subplots(nrows = 1, ncols = 2, figsize = (16,8))

        axarr[0].plot(x,StdForMonths)
        axarr[0].grid(linestyle = 'dashed')
        axarr[0].set_xlabel('Month', fontsize=13)
        axarr[0].set_ylabel('Monthly Std', fontsize= 13)
        axarr[0].set_title('The Standard Devation for Each Month for each Month Over the 136 Years')


        axarr[1].grid(linestyle='dashed')
        axarr[1].set_xlabel('Year', fontsize=13)
        axarr[1].set_ylabel('Yearly Std', fontsize= 13)
        axarr[1].set_title('The Standard Deviation for Each Year Over the 136 Years')
        axarr[1].plot(f,StdForYears)
```

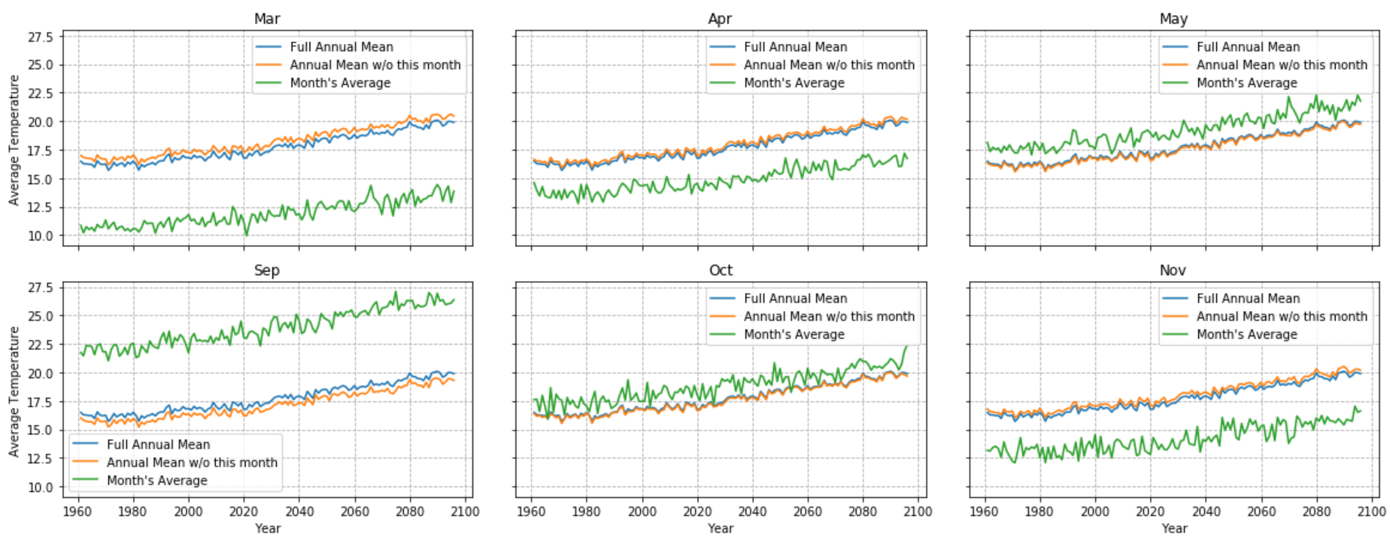
```
Out[9]: [<matplotlib.lines.Line2D at 0x11f251da0>]
```



Exercise 2.3: Prepare a plot with six panels (arranged as you wish) which depicts the annual dependence of the average monthly temperature for meteorological Spring (Mar, Apr, May) and Fall (Sep, Nov, Dec) seasons

In [10]:

```
'''
Alot prettier chunk of code that performs the same tasks as above
'''

fig, axarr = plt.subplots(nrows=2, ncols=3, figsize=(16,6), sharex=True, sharey=True)
fig.tight_layout(h_pad=2, w_pad=2)



# axis labels outside of loops
axarr[0][0].set_ylabel("Average Temperature")
axarr[1][0].set_ylabel("Average Temperature")
axarr[1][0].set_xlabel("Year")
axarr[1][1].set_xlabel("Year")
axarr[1][2].set_xlabel("Year")

# For the Spring Months:
for m, monthName in enumerate(["Mar", "Apr", "May"]):
    axarr[0][m].set_title(monthName)
    axarr[0][m].plot(df.mean(axis=1), label="Full Annual Mean")
    axarr[0][m].plot(df.drop('t'+str(m+3), axis=1).mean(axis=1), \
                     label="Annual Mean w/o this month")
    axarr[0][m].plot(df['t'+str(m+3)], label="Month's Average")
    axarr[0][m].legend()
    axarr[0][m].grid(linestyle='dashed')

# For the Fall Months:
for m, monthName in enumerate(["Sep", "Oct", "Nov"]):
    axarr[1][m].set_title(monthName)
    axarr[1][m].plot(df.mean(axis=1), label="Full Annual Mean")
    axarr[1][m].plot(df.drop('t'+str(m+9), axis=1).mean(axis=1), \
                     label="Annual Mean w/o this month")
    axarr[1][m].plot(df['t'+str(m+9)], label="Month's Average")
    axarr[1][m].legend()
    axarr[1][m].grid(linestyle='dashed')
```



Exercise 3

Exercise 3.1: Prepare a loop such that given when it is given as an input a string variable it produces as output another string variable equal to the first string in reversed order, e.g. a = "abcd" and reverseda = "dcba"

```
In [11]: def reverse(a):
             '''
             I create a reverse function incase it is needed for later..
             this function reads through any string when put in a = ''
             and then will create a new string where it will append the read character
             to the list so its like appending a new value to the start of a list rather
             than the end.
             '''
             Reverseda = ''
             for char in a:

                 Reverseda = char + Reverseda

             return Reverseda
             print(Reverseda)
```

```
In [12]: a = 'abcd'
         reverse(a)

Out[12]: 'dcba'
```

Exercise 3.2: Use a loop to convert the string "Testing loops and strings..." into another string, changing spaces into "_" (underscore).

```
In [13]: test = 'Testing loops and Strings'
         reversedtest = ''
         for char in test:
             if char == ' ':
                 reversedtest = reversedtest + '_'
             else:
                 reversedtest = reversedtest + char

         print(reversedtest)

         Testing_loops_and_Strings
```

Exercise 3.3: Given the list AA = [1.0, -2.0, 3.0, 5.5, 0.3] and considering that these are the values of the coefficients for a polynomial Px = AA[0] + AA[1]x + ... + AA[n]x**n, prepare a loop that computes the polynomial value for a given independent variable x value. For example, in the given case Px = 7.8 for x = 1

```
In [14]: '''
         We Know:
             AA = [1.0,-2.0,3.0,5.5,0.3]
             x = input()
             P_x = AA[0]+AA[1]x+...+AA[n]x**n

         '''
         def functionPoly(x):
             AA = [1.0,-2.0,3.0,5.5,0.3]
             Plist = []

             for n in range(5):
                 P_x = AA[n]*x**n
                 Plist.append(P_x)

             #print(Plist) #unhashtag this line to see the sum of each intermediate step, which shows the a
         ffect
                                     # of that polynomial term
             print(np.sum(Plist))
```

```
In [15]: functionPoly(1)

         7.8
```

Exercise 3.4: Compute for the loaded temperature dataset the average seasonal temperatures and gives as a results a Python list with these temperatures

In [16]:
```
def SeasonalAvgTemp():

    list1 = []
    SeasonalTemp = []
    Months = ['January','February','March','April','May','June','July','August',\
           'September','October','November','December']
    for m, monthname in enumerate(Months):
        MonthMean = '%.2f'%np.mean(df['t'+str(m+1)])
        list1.append(float(MonthMean))

    WintAvg = (list1[11]+list1[0]+list1[1])/3
    SeasonalTemp.append(float('%.2f'%WintAvg))
    SpringAvg =(list1[2]+list1[3]+list1[4])/3
    SeasonalTemp.append(float('%.2f'%SpringAvg))
    SummerAvg =(list1[5]+list1[6]+list1[7])/3
    SeasonalTemp.append(float('%.2f'%SummerAvg))
    AutumnAvg =(list1[8]+list1[9]+list1[10])/3
    SeasonalTemp.append(float('%.2f'%AutumnAvg))


    print(SeasonalTemp)
```

In [17]:
```
SeasonalAvgTemp()
#this list is Winter, Spring, Summer, August
```

```
[10.07, 15.34, 26.44, 18.98]
```

Exercise 3.5: Build a code that, for a given month and considering the loaded temperature dataset, finds the mean and minimum temperature values for the selected month and then it builds a Python list with the years that have an average temperature for the selected month that is less than the average value of the minimum and mean temperature

In [18]:
```
def MonthMinMean():

    List123 = [] #list of each months min and mean values
    List321 = [] #List of each months average value of min and mean values
    input1 = str(input())# input
    Months = ['January','February','March','April','May','June','July','August',\
          'September','October','November','December']
    AR = [] #list for months that have less than the average value of min and mean temperatures
    Years = list(range(1961,2097))



    for m, MonthName in enumerate (Months):
        '''
        Evaluates the min, mean and the average of the two to append to a list for each month
        '''
        MonthMin = np.min(df['t'+str(m+1)])
        MonthMean = float('%.2f'%np.mean(df['t'+str(m+1)]))
        element = MonthMin, MonthMean
        List123.append(element)

        MinMeanAvg = (MonthMin+MonthMean)/2
        List321.append(MinMeanAvg)

    if str(input1) in Months:
        '''
        Takes the user input for what month they want to evaluate
        '''
        mn = Months.index(str(input1))
        lp = df['t'+str(mn+1)]


        for y,n in enumerate(lp):
            '''
            Appends years where the months are lower than the average of the min and mean
            '''
            if n < List321[mn]:
                AR.append(y+1961)


    print(AR)
```

In [19]:
```
MonthMinMean()
```

```
June
[1961, 1962, 1963, 1964, 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1974, 1975, 1976, 1979, 1981,
1982, 1983, 1984, 1985, 1986, 2001, 2006, 2020, 2023]
```

Exercise 4

Exercise 4.1: Prepare a file with a function called fence such that given two strings as arguments: string_1 = "aaa" and string_2 = "bbb", the output is aaa_bbb_aaa. In the same file define a second function called outer such that given a string returns another string made up of just the first and last characters of its input. Therefore if the input is Betis the function output should be Bs. Include in both cases a docstring with a brief function description and an example. Load the functions from the file and check what is the output of this statement: print outer(fence('carbon', '+')).

In [20]:
```
def fence(arg1, arg2):
    '''
    this function takes two arguments, combines the stamements into the
    required output and then prints it
    '''
    fencestatement = arg1+'_'+arg2+'_'+arg1
    #print(fencestatement)
    return fencestatement

def outer(function, pos1,pos2):
    '''
    This function takes the function of fence and prints the first and last letters of the doc str
ing
    produced by fence so for fence(carbon, +) we expect to see 'cn'
    '''
    out = fence(pos1,pos2)
    print(str(out[0])+str(out[-1]))
```

In [21]:
```
outer(fence,'carbon','+')
```

```
cn
```

Exercise 4.2: Gaussian distributed data are frequently normalized to have a mean value equal to zero and a standard deviation equal to one substracting the actual mean value and dividing by the standard deviation of the dataset. Making use of the mean and std NumPy methods, define a function that takes as an argument a data vector, a new mean value, and a new standard deviation value and transforms the original set of data to a new set with a the new mean as its average value and with a dispersion given by the new standard deviation value. By default the function should standardize the data to mean = 0 and sdev = 1.

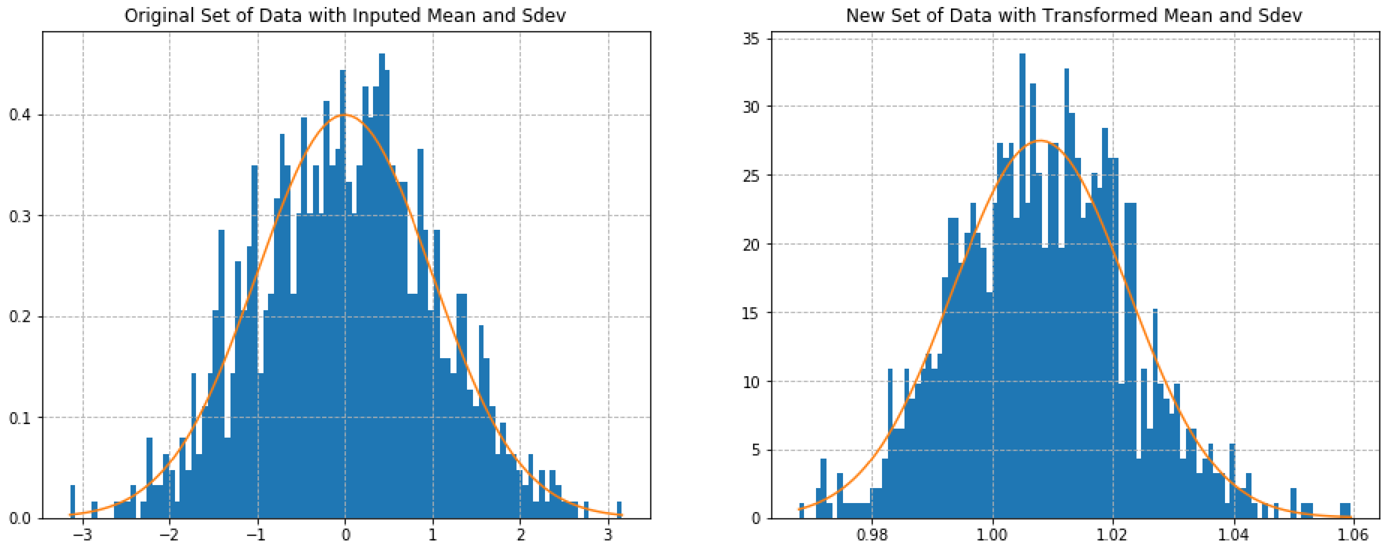
The gaussian distribution equation is given as:

$$P(x) = \frac{1}{\sigma\sqrt{(2\pi)}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Here we know σ is denoting sdev, μ is denoting mean and x is the mean of the data set

In [22]:
```
def GuassianDist(datavector,mean=0,sdev=1):

    fig, axarr = plt.subplots(nrows=1, ncols=2, figsize=(16,6))
    count, bins, ignored = axarr[0].hist(datavector, 100, density=True)
    P = 1/(sdev * np.sqrt(2 * np.pi)) * np.exp( - (bins - mean)**2 / (2 * sdev**2) )
    axarr[0].plot(bins,P)
    axarr[0].grid(linestyle='dashed')
    axarr[0].set_title('Original Set of Data with Inputed Mean and Sdev')

    '''
    now to make new data, based off a linear transform; y=mx+b, where x is the random data, Y is t
he
    new Data, m and b are the adjusting factors
    '''
    m = np.average(datavector)
    b = np.std(datavector)

    new_mean = m*mean+b
    new_std = np.abs(m*sdev)

    newcount, newbins, ignored = axarr[1].hist(np.random.normal(new_mean,new_std,1000),100, densit
y=True)
    P1 = 1/(new_std * np.sqrt(2 * np.pi)) * np.exp( - (newbins - new_mean)**2 / (2 * new_std**2) )
    axarr[1].plot(newbins,P1)
    axarr[1].set_title('New Set of Data with Transformed Mean and Sdev')
    axarr[1].grid(linestyle = 'dashed')
```

In [23]:
```
GuassianDist(np.random.normal(0,1,1000))
```

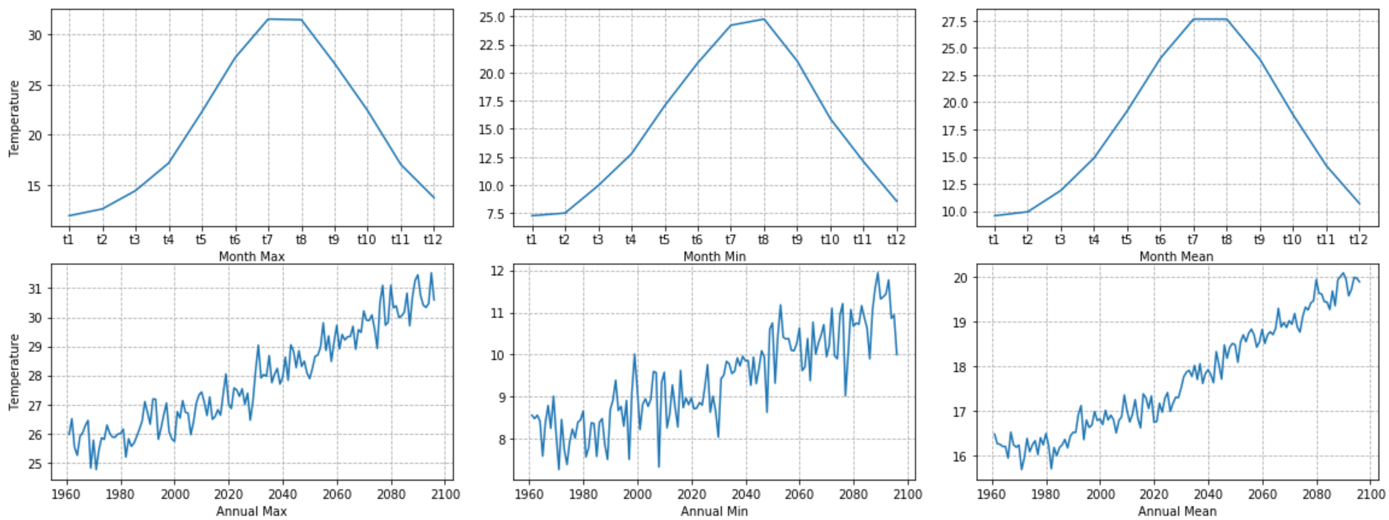


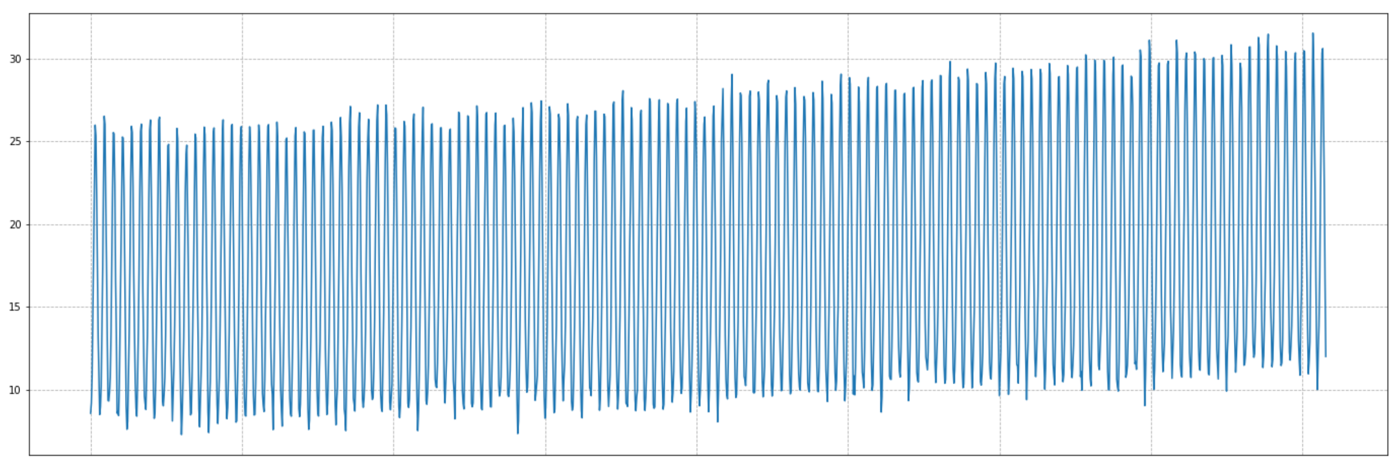
Exercise 4.3: Define a function that reads out temperature data from the sample Cyprus dataset and prepare graphics. Prepare a function with helpful docstring and comments that for given list of file names prepares a plot with three columns for each data file: the first including the max, min and mean monthly temperatures, the second the max, min, and mean annual temperatures, and the third depicting the monthly temperatures for all years.

In [24]:
```python
def Monthtemps(filename): #use T_Agrinio_EM
    '''
    This function creates three seperate plots;
                1. Max, Min and Mean of monthly temperautes
                2. Max, min and mean of annual temperatues
                3. all months temps for all years
    it acquires the data for this from the file used in exercies 1 and 2
    '''
    #read the filename
    df = pd.read_csv((filename)+'.csv',header = 0, index_col = 0)
    df.head()


    Months = ['January','February','March','April','May','June','July','August',\
          'September','October','November','December']


    #this block of code creates the array of graphs with the required columns columns
    fig, axarr = plt.subplots(nrows=2, ncols=3, figsize=(16,6), sharex=False, sharey=False)
    fig.tight_layout(h_pad=1, w_pad=2)
    #Once you figure out what is meant by monthly temperatures for all years can change nrows = 3

    '''
    Setting up parameters for the requested plots
    '''
    #First row
    axarr[0][0].set_ylabel('Temperature')
    axarr[0][0].set_xlabel('Month Max')
    axarr[0][1].set_xlabel('Month Min')
    axarr[0][2].set_xlabel('Month Mean')

    #second row
    axarr[1][0].set_ylabel('Temperature')
    axarr[1][0].set_xlabel('Annual Max')
    axarr[1][1].set_xlabel('Annual Min')
    axarr[1][2].set_xlabel('Annual Mean')

    '''
    Loop to Evaluate and plot the requested problem
    '''
    for m in range(3):
        if m == 0:
            #maximum
            axarr[0][m].plot(np.max(df))
            axarr[1][m].plot(np.max(df.T))
        elif m ==1:
            #minumum
            axarr[0][m].plot(np.min(df))
            axarr[1][m].plot(np.min(df.T))
        else:
            #mean
            axarr[0][m].plot(np.mean(df))
            axarr[1][m].plot(np.mean(df.T))

        axarr[0][m].grid(linestyle='dashed')  #grids to make plots look nicer
        axarr[1][m].grid(linestyle='dashed')


    '''
    I think this is what is asked for the monthly temperatures for all years? I'm not sure... I do
n't know if it's
    asking for this or if it's asking for the max, min and mean of month temperatues of all years
 which would work
    in the axarr method as it's asking for 3 more plots so it keeps the 3x3 shape and fills all. w
hile if I add
    the code below to the axarr loop, it would produce the same plot in [3][0] and leave the other
2 empty and would
    be very hard to see the graph due to sharing the width with the other two plots... the plot is
hard to see
    when it takes up a full line
    '''
```

```
    data = []
    data2 = []

    for m in range(1961,2097):
        data.append(df.T[m])

    for n in range(136):
        for l in range(12):
            data2.append(data[n][l])

    plt.figure(figsize=(24,8))
    plt.plot(data2)
    plt.grid(linestyle='dashed')
    '''
    the x axis is wack for this graph as a result of plotting a list, where the zero position is j
anuary 1961, fist
    position is february 1961, so the x axis is zero to 136*12 not the years
    '''
    plt.tick_params(axis='x', which='both', bottom=False, top=False, labelbottom=False)
```

In [25]:
```
Monthtemps('T_Agrinio_EM')
```





Exercise 4.4: Write a function that generates a random password. The password should have a random length of between 10 and 12 random characters from positions 33 to 122 in the ASCII table. Your function will not take any parameters and will return the password as its only result. Make another function that checks if the password has at least two lowcase, two uppercase, and two digit characters and check how many times you need to run the original function to obtain a compliant password.

In [26]:
```
def randpass():
    randpass1 = ''

    asc = []
    for n in range(0,int(np.random.random_integers(10,12,1))):
        asc.append(int(np.random.random_integers(33,122,1)))

    for n in asc:
        randpass1 = randpass1+str(chr(n))
    return(randpass1)


def passwordcheck(randpass1,counts=0):
    l,u,d, count = 0,0,0,counts
    lowercase = []
    uppercase = []
    twodigits = []
    for char in randpass1:
        lowercase.append(char.islower())
        uppercase.append(char.isupper())
        twodigits.append(char.isdigit())

    for m in range(0,len(lowercase)):
        if lowercase[m] == True:
            l = l+1
    for m in range(0,len(uppercase)):
        if uppercase[m] == True:
            u = u+1
    for m in range(0,len(twodigits)):
        if twodigits[m] == True:
            d = d+1
    if l&u&d >= 2:
        print(randpass1,'is good and took:',counts,' randomisations until this password is achieve
d')
    else:
        newpassword = randpass()
        count = count +1
        passwordcheck(newpassword,count)
```

In [27]:
```
passwordcheck(randpass())
```

```
77-QkKzSKC,R is good and took: 31  randomisations until this password is achieved
```

Exercise 5

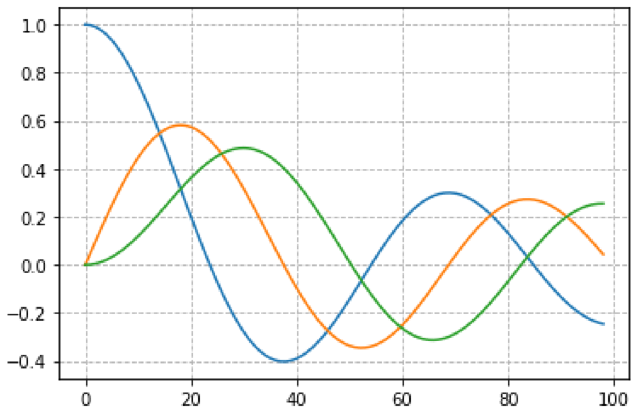
Exercise 5.1: The NIST Digital Library of Mathematical Functions (DLMF) is a very useful site, where you can find an updated and expanded version of the well-known reference \texit{Handbook of Mathematical Functions} of Abramowitz and Stegun. Define a function to compute the Bessel function of the first kind of integer index from the series 10.2.2 in the DLMF, add a docscript and plot the functions of order 0, 1, and 2 in the interval of x between 0 and 10.

The bessel function in question is:

$$J_\upsilon(x) = (\frac{1}{2}x)^\upsilon \sum_{k=0}^{\infty}(-1)^k \frac{(\frac{1}{4}x^2)^k}{k!\Gamma(\upsilon + k + 1)}$$

In [28]:
```
'''
want a function to evaluate this function for different v for 0,1 and 2 and x being in the interva
l of 0 and 10
'''

def BesselFunction(x,v,k):
    J = 0
    coeff = (0.5*x)**v

    for k in range(k):
        J += (-1)**k*(0.25*x*x)**k/(math.factorial(k)*(math.gamma(v+k+1)))

    return coeff*J
```

In [29]:
```
for n in range(3):
    v0 = BesselFunction(x=np.linspace(0,10,99),v=n,k=99)
    plt.plot(v0)
    plt.grid(linestyle = 'dashed')
```

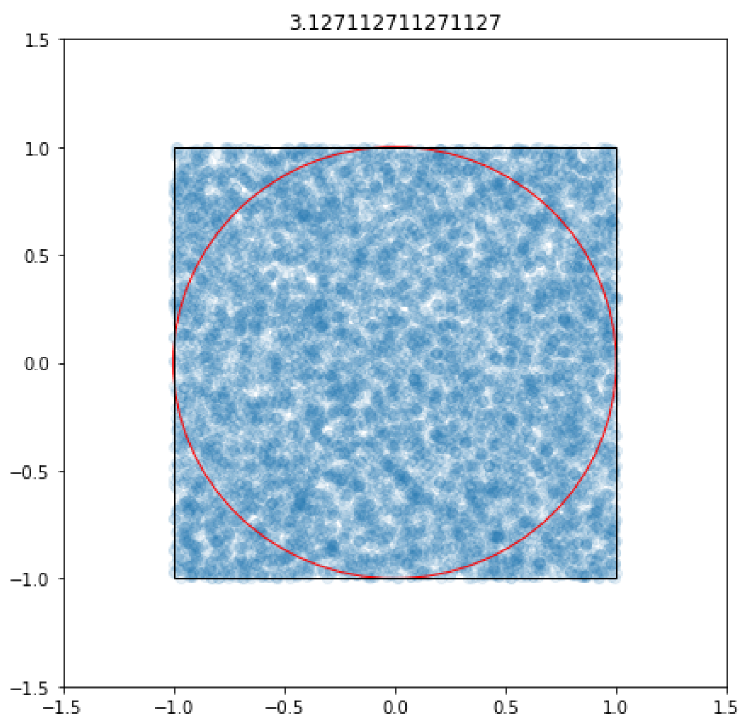


Exercise 5.2: Define and test a function that estimates the value of the special constant pi by generating N pairs of random numbers in the interval -1 and 1 and checking how many of the generated number fall into a circumference of radius 1 and centered in the origin. Improve the function showing in a graphical output the square, the circumference and the points inside and outside the circumference with different colors.

In [30]:
```
def PiEstimate(nPairs):
    test_points = []
    in_circle = []

    for point in range(nPairs):
        #get test point coords
        x = random.uniform(-1,1)
        y = random.uniform(-1,1)
        test_points.append((x,y))

        if (np.sqrt(x*x+y*y)) <=1:
            in_circle.append((x,y))

        pi = 4*len(in_circle)/nPairs

    return test_points, pi
```

In [31]:
```python
'''Circle area: pi r^2 = pi
So square area: 4r^2 = 4
Use NM Rule and count how many in circle vs total (in square)
so n_circle/n_square = pi/4 thus 4*n_circle/n_square = pi
'''

points, pi = PiEstimate(9999)
points_plot = list(zip(*points))

fig = plt.figure(figsize=(7,7))
ax = plt.gca()

plt.scatter(points_plot[0], points_plot[1], alpha = 0.1)
circle = plt.Circle((0,0),radius = 1, fill = False, ec ='r')
ax.add_artist(circle)
square = plt.Rectangle((-1,-1),2,2,fill = False)
ax.add_artist(square)

ax.set_xlim(-1.5,1.5)
ax.set_ylim(-1.5,1.5)
ax.set_title(pi,fontsize=12)
```

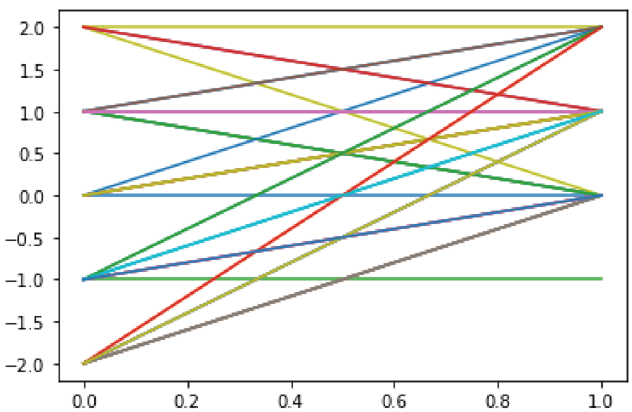
Out[31]:
```
Text(0.5, 1.0, '3.127112711271127')
```



Exercise 5.3: The aim of this exercise is to generate a set of two-dimensional random walks, plot their trajectories and look and the end point distribution. The random walks considered always begin at the origin and take Nstep random steps of unit or zero size in both directions in the x and y axis. For a total number of Nw walks: 1. Compute the trajectories and save the final point of all them. 2. Plot a sample of these random walks in the plane. 3. Plot all the final points together. 4. Compute the average final distance from the origin of the system. 5. Plot a histogram with the values of the distance to the origin.

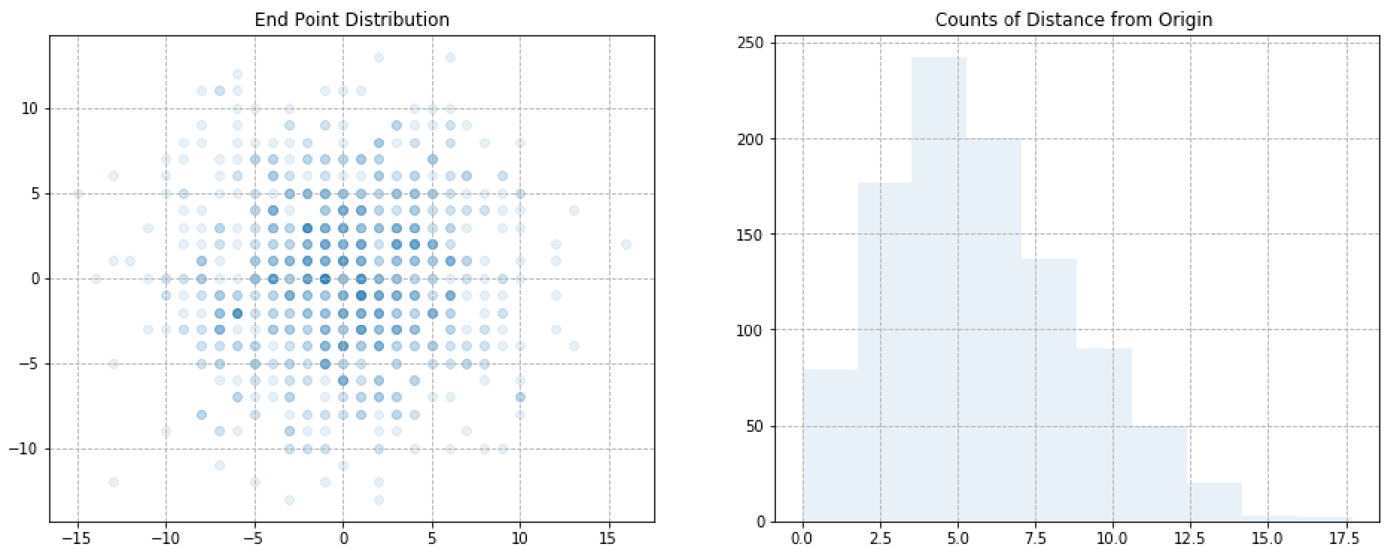
In [32]:
```python
#PART 1
def randwalk(nsteps):
    '''
    takes a random walk in nsteps
    '''
    x = 0
    y = 0
    path = [(0,0)]
    for n in range(nsteps):
        #to randomly decide where to go
        direc = random.choice([(0,1),(0,-1),(-1,0),(1,0),(0,0)])
        x += direc[0]
        y += direc[1]
        path.append((x,y))

    return path
```

In [33]:
```python
#Part 2
allwalks = []
Nw = 1000
for walk in range(Nw):
    allwalks.append(randwalk(50))
#Example of walk path
for i in range(1):
    plot_xy_lists = list(zip(*allwalks[i]))

    plt.plot(plot_xy_lists)
```



In [34]:
```
#Part 3 & 4
final_points = []
origindist = []
fig, axarr = plt.subplots(nrows=1, ncols=2, figsize=(16,6))


for walk in allwalks:
    '''
    Loop to plot the end point distribution and the distance from the origin
    '''
    final_points.append(walk[-1]) #Determines the final path
    origindist.append(np.linalg.norm(walk[-1])) #Calculates the distance from the origin (sqrt(x^2
+y^2)

end_points_plot = list(zip(*final_points))

axarr[0].scatter(end_points_plot[0],end_points_plot[1],alpha = 0.1)
axarr[0].grid(linestyle = 'dashed')
axarr[0].set_title("End Point Distribution")
axarr[1].hist(origindist, alpha = 0.1)
axarr[1].grid(linestyle = 'dashed')
axarr[1].set_title('Counts of Distance from Origin ')
```

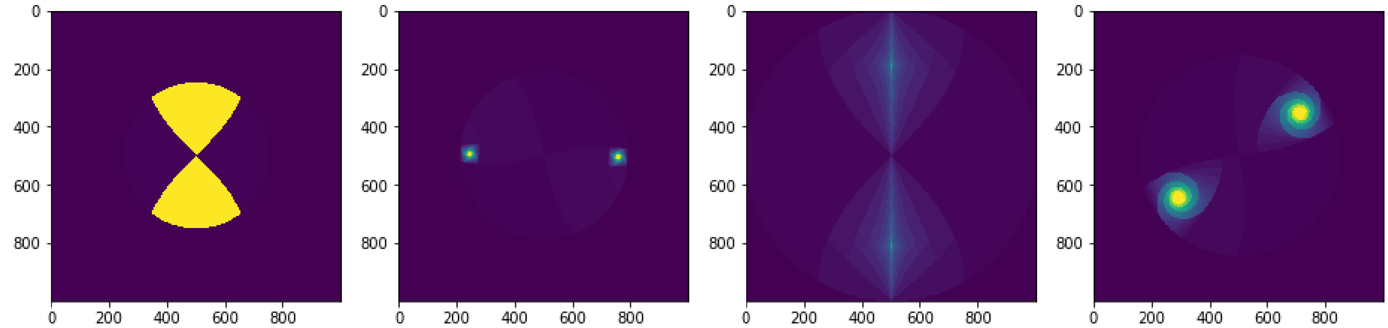
Out[34]:
```
Text(0.5, 1.0, 'Counts of Distance from Origin ')
```



Exercise 5.4: The Julia set is an important concept in fractal theory. Given a complex number a, a point z in the complex plane is said to be in the filled-in Julia set of a function f(z) = z² + a if the iteration of the function over the point does not finish with the point going to infinity. It can be proved that if at some iterate of a point under f(z) the result has a module larger than 2 and larger than the module of a, this point will finish going to infinity. Build and plot the filled-in Julia sets for f(z) with a = (-0.5,0),(0.25,-0.52), (-1,0), (-0.2, 0.66) in the interval of -1 < Re(z), Im(z) < 1 and consider that the point belongs to the set once the previous condition has not been accomplished after Niter = 100. Hint: You can make use of the NumPy meshgrid and the PyPlot pplot functions for displaying the filled-in Julia sets.

In [45]:
```python
'''
Things you need in function:
    a is complex number and z is a point in complex plane
    1.Input: pairs of complex values of a and range for z, which forms nxn grid
    2. The equation: z**2 +a
    3. iterate it over a meshgrid(values of z) and if it does't go to inf (module > 2 & > a.... Wh
ile or If ?)
    then it is not in Julia
                                                                              [What is a module again - is it s
qrt(x^2+y^2)]
    4. perform iteration over Niter = 100? - while loop
    5. How do you make it so if it's not in the set be a colour on the pyplot...

Outcome: 4 plots for the different a
'''

def juliaset(Niter):

    #set up parameters
    counter = 100
    z = np.zeros((1000,1000))
    x,y = np.meshgrid(np.linspace(-1,1,1000),np.linspace(-1,1,1000))
    a = [complex(-0.5,0),complex(0.25,-0.52),complex(-1,0),complex(-0.2,0.66)]
    fig, axarr = plt.subplots(nrows=1, ncols=4, figsize=(16,6))

    for n, cmplx in enumerate(a):
        '''
        loop to acqurie the real and complex values for each complex and the module
        '''

        a_re = cmplx.real
        a_im = cmplx.imag
        amodule = np.sqrt(a_re*a_re+a_im*a_im)

        for ll in range(len(x)):
            for mm in range(len(y)):

                '''
                double loop to evaluate every point along the meshgrid and evaluate if the point i
s in the
                julia set through a while loop
                '''
                l = ll/1000*2-1
                m = mm/1000*2-1
                point = complex(l,m)

                counter = 0
                while abs(point) < 2 and abs(point) < amodule and counter <100:
                    point = point**2+cmplx
                    counter += 1

                ratio = counter/100
                z[ll,mm] = ratio
        axarr[n].imshow(z)
```

In [46]:
```
juliaset(100)
```



Exercise 6

Exercise 6.1: Read the different files with Cyprus towns temperatures provided in the TData folder and build a dataframe combining all the information. The columns should be the year and months and you can distinguish between data for the different towns adding an extra column with the town name.

Hint: the function concat can be very useful in this case.

In [37]:
```
Locations = ['Agrinio', 'Alexandroupolis', 'Alicante', 'Antalya', 'Araxos']

#for n, name in enumerate(Locations):
 #   df+str(n+1) = pd.read_csv('T_'+str(name)+'_EM.csv', header =0, index_col = 0)

df1 = pd.read_csv('T_Agrinio_EM.csv',header = 0, index_col = 0)
df2 = pd.read_csv('T_Alexandroupolis_EM.csv',header = 0, index_col = 0)
df3 = pd.read_csv('T_Alicante_EM.csv',header = 0, index_col = 0)
df4 = pd.read_csv('T_Antalya_EM.csv',header = 0, index_col = 0)
df5 = pd.read_csv('T_Araxos_EM.csv',header = 0, index_col = 0)

totaldata = pd.concat([df1,df2,df3,df4,df5], keys = ['Agrinio', 'Alexandroupolis', 'Alicante', 'An
talya', 'Araxos'])
totaldata.columns = ['January','February','March','April','May','June',
                     'July', 'August', 'September','October','November','December']
'''
To see if they did combine will see the first 137 rows since it should show 2096, for Agrinio and
 then
year 1961 for Alexandroupolis
'''
totaldata.head(137)
```

Out[37]:

		January	February	March	April	May	June	July	August	September	October	November	December
	year												
Agrinio	1961	8.56	9.18	10.86	14.62	18.13	21.98	25.97	25.33	21.75	17.61	13.15	10.70
	1962	8.48	8.94	10.21	13.93	17.34	22.34	26.51	26.01	21.45	17.65	13.10	9.33
	1963	9.30	9.85	10.75	13.45	17.71	22.24	25.52	25.44	22.35	16.57	13.41	8.56
	1964	8.67	8.42	10.49	14.27	17.52	21.87	25.25	25.19	22.32	17.99	13.52	9.03
	1965	7.59	8.05	10.68	13.45	17.64	22.75	25.90	25.53	22.26	16.99	13.31	10.42
	...	...	...	...	...	...	...	...	...	...	...	...	...
	2093	11.77	12.62	13.84	16.02	21.69	25.75	29.75	30.34	25.94	20.21	15.72	12.95
	2094	11.74	10.86	14.31	16.07	21.39	27.20	30.47	30.39	26.01	20.71	17.04	13.71
	2095	10.94	11.73	12.86	17.19	22.29	26.55	31.53	29.83	26.08	21.86	16.50	12.29
	2096	10.00	12.20	13.87	16.73	21.76	25.84	30.33	30.60	26.37	22.40	16.62	11.99
Alexandroupolis	1961	5.41	6.40	8.75	13.61	17.28	21.87	25.97	24.51	20.25	14.78	10.62	7.86

137 rows × 12 columns

Exercise 6.2: A different way to combine the Cyprus towns temperature data provided in the TData folder is to build a dataframe whose index is a hierarchical one, with the year and month and there are as many columns as towns, labeled with the

town names.

Hint: The concat and unstack functions can be helpful in this

In [38]:
```
pd.DataFrame.unstack(totaldata)
```

Out[38]:

	January										...	December						
year	1961	1962	1963	1964	1965	1966	1967	1968	1969	1970	...	2087	2088	2089	2090	2091	2092	2(
Agrinio	8.56	8.48	9.30	8.67	7.59	8.86	9.16	8.25	9.01	8.09	...	11.46	12.76	12.61	13.74	12.56	12.26	1:
Alexandroupolis	5.41	4.49	5.86	5.51	4.48	5.55	6.47	4.54	5.57	4.91	...	7.51	10.06	9.60	10.84	9.30	8.93	!
Alicante	11.55	10.72	11.04	11.29	10.74	10.63	11.66	11.00	10.28	10.41	...	14.44	14.30	14.55	15.36	14.81	15.00	1!
Antalya	9.63	9.41	9.75	9.77	9.03	10.28	10.29	9.66	9.37	9.43	...	13.55	14.32	13.99	14.97	14.16	13.37	1:
Araxos	10.32	10.17	10.96	10.25	9.30	10.53	10.90	9.88	10.69	9.83	...	13.12	14.46	14.39	15.34	14.31	14.07	1

5 rows × 1632 columns

Exercise 6.3: Compute the correlation matrix between the temperatures in the provided

Cyprus cities from the previous exercise dataframe.

In [39]:
```
pd.DataFrame.corr(totaldata)
```

Out[39]:

	January	February	March	April	May	June	July	August	September	October	November	Decer
January	1.000000	0.935128	0.903183	0.788183	0.508684	0.262493	0.238966	0.373861	0.663833	0.815572	0.891838	0.88
February	0.935128	1.000000	0.938298	0.818324	0.544468	0.301938	0.263190	0.391466	0.686567	0.833171	0.892563	0.85
March	0.903183	0.938298	1.000000	0.891873	0.658987	0.435398	0.381539	0.508778	0.769744	0.878337	0.904261	0.84
April	0.788183	0.818324	0.891873	1.000000	0.845778	0.668932	0.619504	0.702199	0.876089	0.906807	0.851131	0.73
May	0.508684	0.544468	0.658987	0.845778	1.000000	0.901518	0.888242	0.910603	0.905837	0.789029	0.642709	0.47
June	0.262493	0.301938	0.435398	0.668932	0.901518	1.000000	0.952397	0.935087	0.820377	0.623037	0.439847	0.26
July	0.238966	0.263190	0.381539	0.619504	0.888242	0.952397	1.000000	0.950431	0.803802	0.588385	0.401430	0.22
August	0.373861	0.391466	0.508778	0.702199	0.910603	0.935087	0.950431	1.000000	0.886231	0.699904	0.536664	0.36
September	0.663833	0.686567	0.769744	0.876089	0.905837	0.820377	0.803802	0.886231	1.000000	0.903181	0.791883	0.63
October	0.815572	0.833171	0.878337	0.906807	0.789029	0.623037	0.588385	0.699904	0.903181	1.000000	0.914714	0.77
November	0.891838	0.892563	0.904261	0.851131	0.642709	0.439847	0.401430	0.536664	0.791883	0.914714	1.000000	0.84
December	0.880541	0.854624	0.845019	0.738539	0.476581	0.267272	0.225486	0.363640	0.636976	0.779918	0.846277	1.00

Exercise 6.4: We provide as an example data set the file 'meteodat.csv' with an excerpt of data with a 10 minute frequency from an automated meteorological station with a span of two months (Jan and Feb 2014). Data are comma separated and you can read them using pd.read_csv. The first column is the date and the second the time. Transform this data into a dataframe with an index of datetime, compute dataframes with downsampling to hourly and daily average values for temperature (Tout), pressure (Pressure), relative humidity (H out), wind speed (Wind Speed), and dew point temperature (Dew point); sum of rainfall (Rain); and maximum and minimum values of temperature (Tmax and Tmin) and show the correlation between these variables at hourly and daily scales.

In [40]:
```
import datetime

odf =  pd.read_csv('meteodat.csv',header = 0, index_col = False)
odf.head()
#odf.head(50000) #To find out there is 8495 rows of trash
```

Out[40]:

	date	time	Tout	Tmax	Tmin	H out	Dew point	Wind Speed	Wind Direction	Wind Speed Hi	...	Wind Chill	Heat Index	THM Index	Pressure	Rain	Rain Max	Tin	H in
0	01/01/14	0:10	15.1	15.1	15.1	85	12.6	9.7	SSW	17.7	...	14.7	15.1	14.7	1023.0	0.0	0.0	16.4	56
1	01/01/14	0:20	15.1	15.1	15.1	85	12.6	8.0	SSW	14.5	...	14.9	15.1	14.9	1022.9	0.0	0.0	16.6	56
2	01/01/14	0:30	15.1	15.1	15.1	86	12.7	8.0	SSW	16.1	...	14.9	15.1	14.9	1022.8	0.0	0.0	16.6	56
3	01/01/14	0:40	14.9	15.1	14.9	86	12.6	8.0	SSW	14.5	...	14.8	14.9	14.8	1022.7	0.0	0.0	16.6	56
4	01/01/14	0:50	14.9	14.9	14.8	88	12.9	6.4	SSW	12.9	...	14.9	14.9	14.9	1022.6	0.0	0.0	16.6	56

5 rows × 21 columns

In [41]:

```python
'''
To be able to downsample into days and hours we need to combine the date and time columns into
a date:time format in the shape d/m/y/h:m and then reappend it to a new table so we can
easily perform the proposed computations
'''
#need to transform the year from /14 to /2014
date = []
for n in range(len(odf)):
    date.append(odf.iloc[n]['date'][0:-2]+str(20)+odf.iloc[n]['date'][-2:])
time = odf['time']

#create new data frame so we can combine date and time easily
dictio = {'date':date,'time':time}
newodf = pd.DataFrame(dictio)
date_time = newodf[['date','time']].agg('/'.join,axis=1)

#Format it into the correct form so we can easily downsample
stamps = []
for n in range(len(date_time)):
    stamps.append(datetime.datetime.strptime(date_time.iloc[n], '%d/%m/%Y/%H:%M'))

index = pd.DatetimeIndex(stamps)
print(type(index))

#new dataset ready for downsampling
new1 = odf.copy()
new1 = new1.set_index(index)
new1= new1.drop(columns = ['date','time'])
new1.head()
```

```
<class 'pandas.core.indexes.datetimes.DatetimeIndex'>
```

Out[41]:

	Tout	Tmax	Tmin	H out	Dew point	Wind Speed	Wind Direction	Wind Speed Hi	Wind Direction Hi	Wind Chill	Heat Index	THM Index	Pressure	Rain	Rain Max	Tin	H in	D P In
2014-01-01 00:10:00	15.1	15.1	15.1	85	12.6	9.7	SSW	17.7	SW	14.7	15.1	14.7	1023.0	0.0	0.0	16.4	56	
2014-01-01 00:20:00	15.1	15.1	15.1	85	12.6	8.0	SSW	14.5	SSW	14.9	15.1	14.9	1022.9	0.0	0.0	16.6	56	
2014-01-01 00:30:00	15.1	15.1	15.1	86	12.7	8.0	SSW	16.1	SW	14.9	15.1	14.9	1022.8	0.0	0.0	16.6	56	
2014-01-01 00:40:00	14.9	15.1	14.9	86	12.6	8.0	SSW	14.5	SSW	14.8	14.9	14.8	1022.7	0.0	0.0	16.6	56	
2014-01-01 00:50:00	14.9	14.9	14.8	88	12.9	6.4	SSW	12.9	SSW	14.9	14.9	14.9	1022.6	0.0	0.0	16.6	56	

In [42]:

```python
#Daily Downsample
Daily = new1.resample('d').agg({'Tout':'mean','Pressure':'mean','H out':'mean','Wind Speed':'mean'
, \
                                 'Dew point':'mean','Rain':'sum','Tmax':'max','Tmin':'min'})
Daily.head()
```

Out[42]:

	Tout	Pressure	H out	Wind Speed	Dew point	Rain	Tmax	Tmin
2014-01-01	14.391608	1020.513287	91.881119	3.664336	13.077622	1.2	15.6	11.7
2014-01-02	15.670139	1019.097917	93.250000	8.112500	14.585417	1.4	16.4	14.3
2014-01-03	16.106250	1021.252778	94.027778	7.278472	15.124306	0.8	18.3	14.8
2014-01-04	15.131250	1022.018750	88.013889	9.936111	13.119444	1.0	17.4	12.2
2014-01-05	11.514583	1025.277083	88.375000	2.171528	9.620139	0.2	16.1	7.3

In [43]:
```
#Hourly Downsample
hourly = new1.resample('H').agg({'Tout':'mean','Pressure':'mean','H out':'mean','Wind Speed':'mea
n', \
                                 'Dew point':'mean','Rain':'sum','Tmax':'max','Tmin':'min'})
hourly.head()
```

Out[43]:

	Tout	Pressure	H out	Wind Speed	Dew point	Rain	Tmax	Tmin
2014-01-01 00:00:00	15.020000	1022.800000	86.000000	8.020000	12.680000	0.0	15.1	14.8
2014-01-01 01:00:00	14.800000	1022.150000	88.833333	5.866667	12.983333	0.4	14.9	14.6
2014-01-01 02:00:00	14.266667	1021.616667	92.833333	4.533333	13.116667	0.4	14.6	14.2
2014-01-01 03:00:00	14.450000	1021.266667	94.166667	4.266667	13.516667	0.2	14.6	14.3
2014-01-01 04:00:00	14.550000	1021.150000	94.833333	6.416667	13.750000	0.2	14.7	14.4

In [44]:
```
'''
To show the correlation between the daily and hourly we just follow similar
steps through 6.1,6.2 & 6.3
'''
total_d = pd.concat([hourly,Daily],keys = ['hourly','daily'])
pd.DataFrame.unstack(total_d)
pd.DataFrame.corr(total_d)
```

Out[44]:

	Tout	Pressure	H out	Wind Speed	Dew point	Rain	Tmax	Tmin
Tout	1.000000	0.055500	-0.214522	0.452574	0.727271	-0.025571	0.964104	0.968241
Pressure	0.055500	1.000000	-0.015641	-0.260977	0.039140	-0.172102	0.057483	0.047483
H out	-0.214522	-0.015641	1.000000	-0.258015	0.511729	0.083671	-0.235816	-0.177183
Wind Speed	0.452574	-0.260977	-0.258015	1.000000	0.219287	0.068920	0.433846	0.439407
Dew point	0.727271	0.039140	0.511729	0.219287	1.000000	0.033165	0.681619	0.724961
Rain	-0.025571	-0.172102	0.083671	0.068920	0.033165	1.000000	0.051132	-0.104424
Tmax	0.964104	0.057483	-0.235816	0.433846	0.681619	0.051132	1.000000	0.873714
Tmin	0.968241	0.047483	-0.177183	0.439407	0.724961	-0.104424	0.873714	1.000000