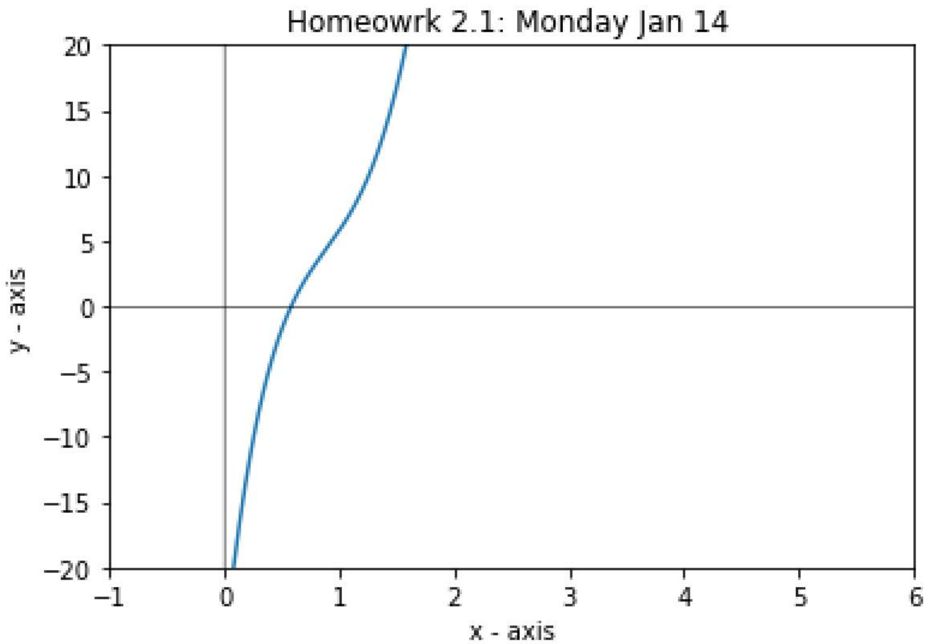
```python
'''
Monday Jan 14 class:
Homework 2.1
Determine the real root off(x) = 0.8x5−8x4+ 46x3−90x2+ 83x−26
(a)  Graphically.
(b)  Using the bisection method to determine the root withεs= 10%.
Employthe initial guesses of xl= 0.5 and xu= 1.0.
(c)  Perform the same computation as in (b) but use the false position methodandεs= 0.2%.
'''

import matplotlib.pyplot as graph
import numpy as np

#Peramiters defined by the user
xL = .05
xU = 1.0
es = .001
iMax = 50

#creating a function class to be called later
def function(x):
    y = .8*x*x*x*x*x - 8*x*x*x*x + 46*x*x*x - 90*x*x + 83*x - 26 #often python struggles tc
    return y

'''
Part A
Graphically
'''

#building a graph
def plotSpace ():
    #imporitng function
    x = np.arange(-10,10,.01)
    y = function(x)
    graph.ylim(-20,20)
    graph.xlim(-1, 6)

    #plotting function
    graph.plot(x, y)

    #setting up graph
    graph.xlabel('x - axis')
    graph.ylabel('y - axis')
    graph.title('Homeowrk 2.1: Monday Jan 14 ')

    #plotting axis
    graph.plot(x, x*0 + 0 , linewidth = .5, color = 'black')
    graph.plot(x*0 +0, y, linewidth =.5, color = 'black')

    #grpahing
    graph.show()

plotSpace()


'''
Part B
```

```
Using the bisection method to determine the root with εs = 10%. Employ
the initial guesses of xl = 0.5 and xu = 1.0.
'''

#buiding the bisection method
def bisectionMethod(xL,xU,es,iMax):
    iCount = 0
    xR = xL
    ea = es
    xOld = None

    for iCount in range(0,iMax):
        xOld = xR
        xR = (xL + xU) / 2 #this averages the function over a range and narrows on the root
        iCount +=1
        if xR != 0:
            ea = abs((xR-xOld)/xR)
        test = function(xL)*function(xR)
        if test < 0:
            xU = xR
        elif test > 0:
            xL = xR
        else :
            ea = 0

        if ea<es or iCount >= iMax:
            break
    print("The numner of iterations was: ", iCount)
    print("Using bisection, the root is: " , xR)
    print("The error was: " , ea*100 , "%")


bisectionMethod(xL,xU,es,iMax)
print("")

'''
Part C
Perform the same computation as in (b) but using the false position method.
'''

def falsePosition(xL,xU,es,iMax):
    iCount = 0
    xR = xL
    ea = es
    xOld = None

    for iCount in range(0,iMax):
        xOld = xR
        xR = xU - (function(xU)*(xL-xU)/(function(xL)-function(xU)))
        iCount +=1
        if xR != 0:
            ea = abs((xR-xOld)/xR)
        test = function(xL)*function(xR)
        if test < 0:
            xU = xR
        elif test > 0:
            xL = xR
```

```python
        else :
            ea = 0
        if ea<es or iCount >= iMax:
            break
    print("The number of iterations was: ", iCount)
    print("Using fasle postion, the root is: ", xR)
    print("The error is: ", ea*100 , "%")

falsePosition(xL,xU,es,iMax)
```

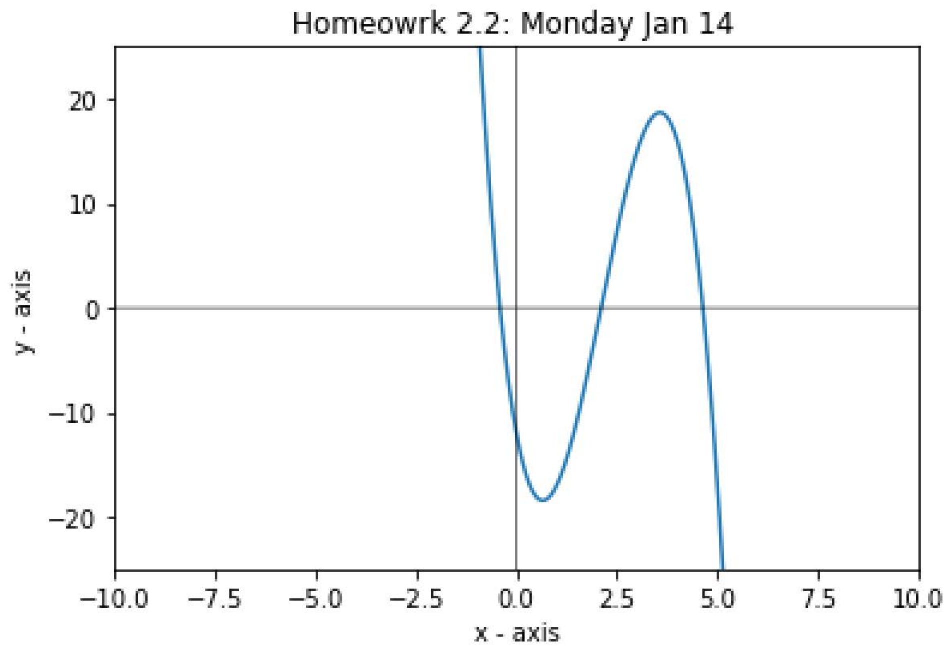



```
The numner of iterations was:  11
Using bisection, the root is:  0.5811279296875002
The error was:  0.0798218711927106 %

The number of iterations was:  11
Using fasle postion, the root is:  0.5811977571089668
The error is:  0.07180703707534478 %
```

```python
'''
Homeowork 2.2
 Determine the lowest real root of f(x) = −3x^3 + 19x^2 − 21x − 12
(a) Graphically.
(b) Using the bisection method to determine the lowest root with εs = 2%.
Employ the initial guesses of xl = −1 and xu = 0.
(c) Perform the same computation as in (b) but using the false position method.
'''

import matplotlib.pyplot as graph
import numpy as np

#peramiters defined by the user
xL = -1.0
xU = 0
es = .002
iMax = 50

def function(x):
    y = -3*x*x*x + 19*x*x -21*x-12
    return y

'''
Part A
Graphically
'''

def plotSpace ():
    #setting up function
    x = np.arange(-10,10,.01)
    y = function(x)
    graph.ylim(-25,25)
    graph.xlim(-10, 10)
    #plotting function
    graph.plot(x, y)

    #setting up graph
    graph.xlabel('x - axis')
    graph.ylabel('y - axis')
    graph.title('Homeowrk 2.2: Monday Jan 14 ')

    #plotting axis
    graph.plot(x, x*0 + 0 , linewidth = .5, color = 'black')
    graph.plot(x*0 +0, y, linewidth =.5, color = 'black')

    #grpahing
    graph.show()

plotSpace()

'''
Part B
Bisection Method
'''

def bisectionMethod(xL,xU,es,iMax):
    iCount = 0
```

```
    xR = xL
    ea = es
    xOld = None

    for iCount in range(0,iMax):
        xOld = xR
        xR = (xL + xU) / 2
        iCount +=1
        if xR != 0:
            ea = abs((xR-xOld)/xR)
        test = function(xL)*function(xR)
        if test < 0:
            xU = xR
        elif test > 0:
            xL = xR
        else :
            ea = 0

        if ea<es or iCount >= iMax:
            break
    print("The numner of iterations was: ", iCount)
    print("The root is: " , xR)
    print("The error was: " , ea*100 , "%")

bisectionMethod(xL,xU,es,iMax)
print("")

'''
Part C
False Postion Method
'''

def falsePosition(xL,xU,es,iMax):
    iCount = 0
    xR = xL
    ea = es
    xOld = None

    for iCount in range(0,iMax):
        xOld = xR
        xR = xU - (function(xU)*(xL-xU)/(function(xL)-function(xU)))
        iCount +=1
        if xR != 0:
            ea = abs((xR-xOld)/xR)
        test = function(xL)*function(xR)
        if test < 0:
            xU = xR
        elif test > 0:
            xL = xR
        else :
            ea = 0

        if ea<es or iCount >= iMax:
            break
    print("The number of iterations was: ", iCount)
    print("The root is: ", xR)
    print("The error is: ", ea*100 , "%")
```

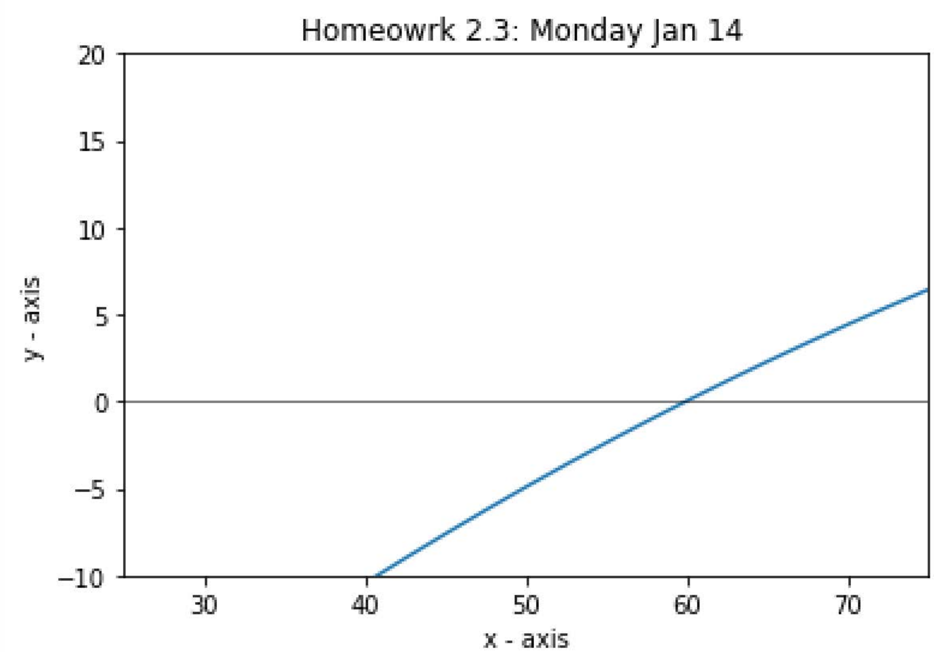
```
falsePosition(xL,xU,es,iMax)
```




```
The numner of iterations was:  11
The root is:  -0.40966796875
The error was:  0.11918951132300357 %

The number of iterations was:  6
The root is:  -0.4095014825799878
The error is:  0.14071685149705257 %
```

```python
'''
Homework 2.3
Textbook problem 5.13.
Velocity given by funcion. Constants are defined. Find mass m.
'''

import numpy as np
import matplotlib.pyplot as graph


#peramiters defined by the user
mass = None
xL = 55
xU = 65
es = .001
iMax = 50

def function(mass):
    y = (9.81 * mass)/15 * (1 - 2.718281828459045235360287**(-(15/mass)*10)) - 36
    return y

def plotSpace ():
    #setting up function
    x = np.arange(-10000,10000,.01)
    y = function(x)
    graph.ylim(-10,20)
    graph.xlim(25,75)
    #plotting function
    graph.plot(x, y)

    #setting up graph
    graph.xlabel('x - axis')
    graph.ylabel('y - axis')
    graph.title('Homeowrk 2.3: Monday Jan 14 ')

    #plotting axis
    graph.plot(x, x*0 + 0 , linewidth = .5, color = 'black')
    graph.plot(x*0 +0, y, linewidth =.5, color = 'black')

    #grpahing
    graph.show()

plotSpace()


def falsePosition(xL,xU,es,iMax):
    iCount = 0
    xR = xL
    ea = es
    xOld = None

    for iCount in range(0,iMax):
        xOld = xR
        xR = xU - (function(xU)*(xL-xU)/(function(xL)-function(xU)))
        iCount +=1
        if xR != 0:
            ea = abs((xR-xOld)/xR)
```

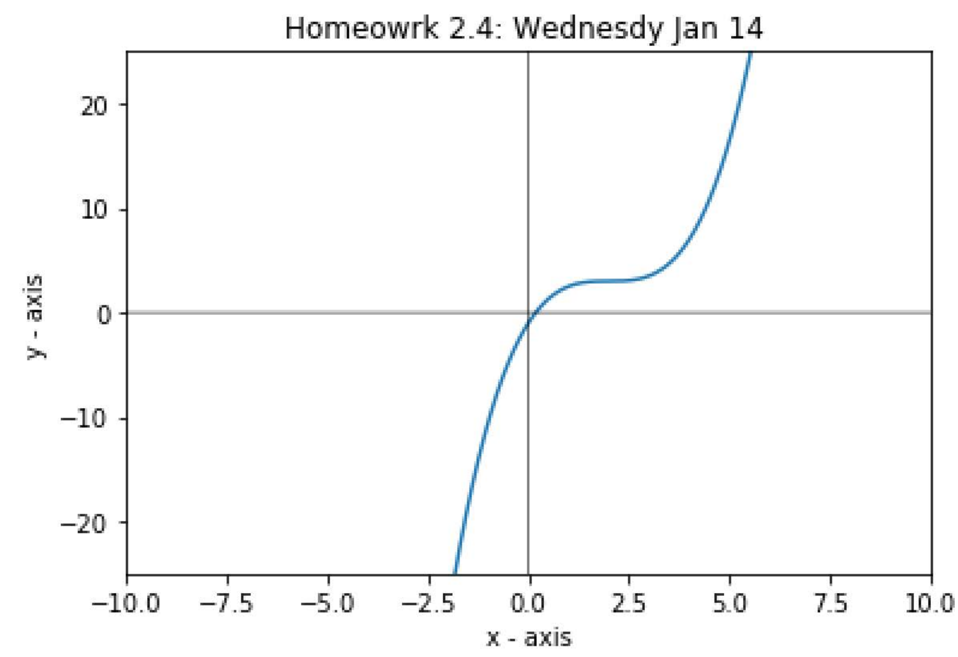
```
        test = function(xL)*function(xR)
        if test < 0:
            xU = xR
        elif test > 0:
            xL = xR
        else :
            ea = 0

        if ea<es or iCount >= iMax:
            break
    print("The number of iterations was: ", iCount)
    print("The root is: ", xR)
    print("The error is: ", ea*100 , "%")


falsePosition(xL,xU,es,iMax)
```




```
The number of iterations was:  3
The root is:  59.95940783769422
The error is:  0.007093037184985085 %
```

```python
'''
Homework 2.4
Determine the real roots of f(x) = 0.5x^3 − 3x^2 + 6x − 1
(a) Graphically.
(b) Using the Newton-Raphson method to within εs = 0.01%.
'''

import numpy as np
import matplotlib.pyplot as graph

#peramiters defined by the user
x = None
x0 = 5
es = .001
iMax = 50


def function(x):
    y = 0.5*x*x*x - 3*x*x + 6*x - 1
    return y

def functionDer(x):
    y = 1.5*x*x - 6*x + 6
    return y

'''
Part A
Graphically
'''

def plotSpace ():
    #setting up function
    x = np.arange(-10,10,.01)
    y = function(x)
    graph.ylim(-25,25)
    graph.xlim(-10, 10)
    #plotting function
    graph.plot(x, y)

    #setting up graph
    graph.xlabel('x - axis')
    graph.ylabel('y - axis')
    graph.title('Homeowrk 2.4: Wednesdy Jan 14 ')

    #plotting axis
    graph.plot(x, x*0 + 0 , linewidth = .5, color = 'black')
    graph.plot(x*0 +0, y, linewidth =.5, color = 'black')

    #grpahing
    graph.show()

plotSpace()

'''
Part B
Newton Raphson
'''
```

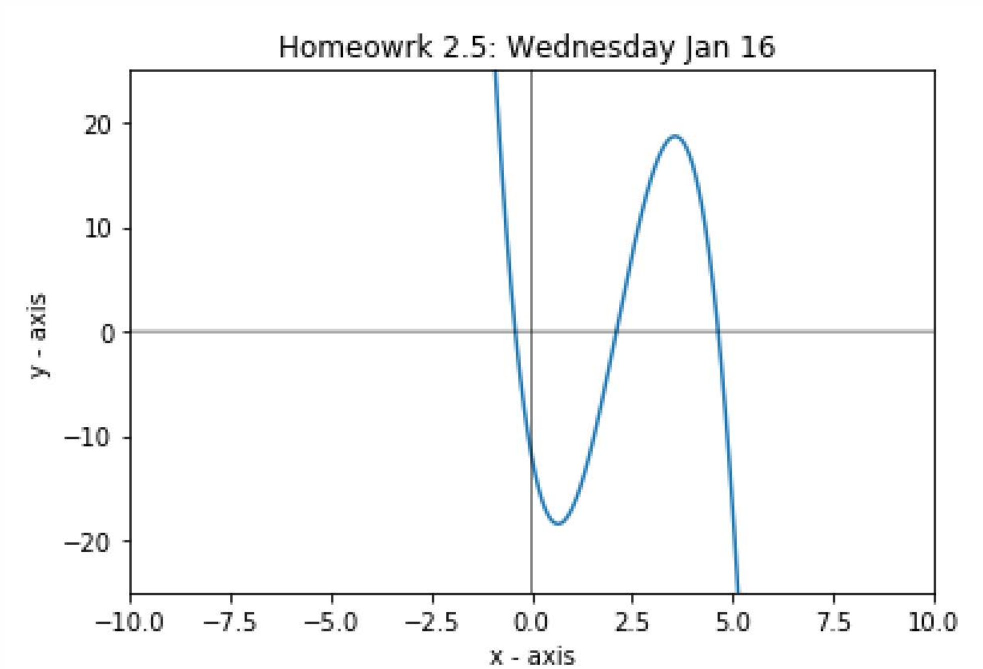
```
def newtonRaphson(x0,es,iMax):
    ea=es
    xR = x0
    iCount = xOld = ea = None
    for iCount in range(0,iMax):
        iCount +=1
        xOld = xR
        xR = xOld - (function(xOld)/functionDer(xOld))
        if xR != 0:
            ea = abs((xR-xOld)/xR)*100
        if ea<es or iCount > iMax:
            break
    print("The number of iterations was ", iCount)
    print("The root of the function is: ", xR)
    print("The error is: ", ea*100 , "%")

newtonRaphson(x0, es, iMax)
```




```
The number of iterations was  10
The root of the function is:  0.18287940716685103
The error is:  0.07405249365191668 %
```

```python
'''
Homework 2.5
Determine all roots of f(x) = −3x^3 + 19x^2 − 21x − 12
(a) Using the Secant method to a value of εs corresponding to three significant
figures.
'''

import numpy as np
import matplotlib.pyplot as graph


#peramiters defined by the user
x = None
iMax = 50
es = .001

#Rroot 1
R1x0 = 3
R1x1 = 6
#root 2
R2x0 = 0
R2x1 = 3
#root 3
R3x0 = -1
R3x1 = 1

def function(x):
    y = -3*x**3 + 19*x**2 - 21*x - 12
    return y


def plotSpace ():
    #setting up function
    x = np.arange(-10,10,.01)
    y = function(x)
    graph.ylim(-25,25)
    graph.xlim(-10, 10)
    #plotting function
    graph.plot(x, y)

    #setting up graph
    graph.xlabel('x - axis')
    graph.ylabel('y - axis')
    graph.title('Homeowrk 2.5: Wednesday Jan 16 ')

    #plotting axis
    graph.plot(x, x*0 + 0 , linewidth = .5, color = 'black')
    graph.plot(x*0 +0, y, linewidth =.5, color = 'black')

    #grpahing
    graph.show()

plotSpace()

def secant(x0,x1,es,iMax):
    ea=es
    iCount = None
```

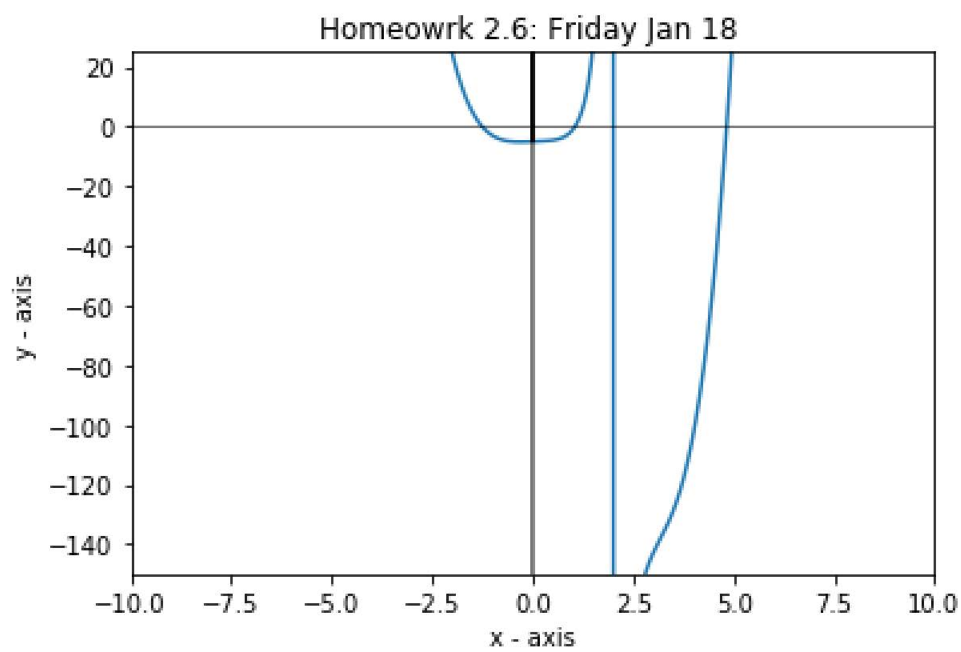
```
    for iCount in range(0,iMax):
        iCount +=1
        xR = x1 - (function(x1)*(x0 - x1))/((function(x0) - function(x1)))
        if xR != 0:
            ea = abs((xR-x1)/xR)*100
        if ea<es or iCount > iMax:
            break
        x0 = x1
        x1 = xR

    print("The number of iterations was ", iCount)
    print("A root of the function is: ", xR)
    print("The error is: ", ea*100 , "%")
    print("")

secant(R1x0, R1x1, es, iMax)
secant(R2x0, R2x1, es, iMax)
secant(R3x0, R3x1, es, iMax)
```




```
The number of iterations was  14
A root of the function is:  4.638186908361574
The error is:  0.007996436000377943 %

The number of iterations was  5
A root of the function is:  2.104866456175301
The error is:  9.221506542952913e-05 %

The number of iterations was  12
A root of the function is:  -0.40972003238293625
The error is:  0.0007932343571165544 %
```

```python
'''
Homework 2.6
Müller’s method. Test it by:
duplicating Example 7.2.
Divide a polynomial f(x) = x**5 - 5*x**4 + x**3 - 6*x +10 by the monomial factor x-2.
'''

import numpy as np
import matplotlib.pyplot as graph
import cmath


#peramiters defined by the user
x = None
iMax = 50
es = .001

#Rroot 1
x0 = 6
x1 = 8
x2 = -5


def function(x):
    y = (x**5 - 5*x**4 + x**3 - 6*x +10) / (x-2)
    return y


def plotSpace ():
    #setting up function
    x = np.arange(-10,10,.01)
    y = function(x)
    graph.ylim(-150,25)
    graph.xlim(-10, 10)
    #plotting function
    graph.plot(x, y)

    #setting up graph
    graph.xlabel('x - axis')
    graph.ylabel('y - axis')
    graph.title('Homeowrk 2.6: Friday Jan 18 ')

    #plotting axis
    graph.plot(x, x*0 + 0 , linewidth = .5, color = 'black')
    graph.plot(x*0 +0, y, linewidth =.5, color = 'black')

    #grpahing
    graph.show()

plotSpace()


def Muller (x0,x1,x2,es,iMax):
    ea = es
    iCount = None
    for iCount in range(0,iMax):
        iCount += 1
```

```
        h0 = x1 - x0
        h1 = x2 - x1
        if h1 ==0:
            break
        d0 = (function(x1)- function(x0))/h0
        d1 = (function(x2)- function(x1))/h1
        if (h1+h0) == 0:
            break
        a = (d1-d0)/(h1+h0)
        b = a*h1 + d1
        c = function(x2)
        rad = cmath.sqrt(b**2 - 4*a*c)
        if abs(b + rad) > abs(b - rad):
            den = b + rad
        else:
            den = b - rad
        xR = x2 + (-2*c)/den
        if xR != 0:
            ea = abs((xR-x2)/xR)*100
        if ea <es or iCount >= iMax:
            exit
        x0 = x1
        x1 = x2
        x2 = xR
    print("A root is: ", xR)
    print("The number of iterations was: ", iCount)
    print("The error is: ", ea*100 , "%")

Muller(x0,x1,x2,es,iMax)
```




```
A root is:  (0.18754828205757743-1.237562118546583j)
The number of iterations was:  18
The error is:  0.0 %
```