Bisection method

```
from math import *

MaxIterations = 1000

print('\n\n*** BISECTION METHOD IMPLEMENTATION ***')

def BisectionMethod(a, b, TOL, N = MaxIterations):

    a = float(a)
    b = float(b)
    i = 1

    while i<=N:

        # Finding Bisection
        m = (a+b)/2

        # Finding Function Value at a b and m
        fa = f(a)
        fm = f(m)
        fb = f(b)

        print(f'iteration = {i:3}',end ="    ")
        print(f'a = {a:15.10f}',end="    ")
        print(f'b = {b:15.10f}',end="    ")
        print(f'm = {m:15.10f}',end="    ")
        print(f"f(m) = {fm:15.10f}")

        # print(f'Iteration-{i} m = {m:3} and f(x2) = {fm:3}')

        if abs(fm) <=TOL or (abs(m-a)) < TOL:
            print(f'\nFinal Root Found')

            print(f'\niteration = {i:3}    a = {a:3.05f}   b = {b:3.06f}   m = {m:3.05f}   f(m) = {fm:3.05f}\n\n')

            # print(f'Iteration-{i}, m = {m:3.10} and f(x2) = {fm:3.10}')

            break
        i = i + 1

        # Checking Interval for next iteration
        if fa*fm>0:
            a = m
        else:
            b = m
    if i > N:
        print("Not Convergent")


def f(x):

    # Input Function
    # g = (x**3) + 4*(x**2) - 10
    g = cos(x) - x*exp(x)
    g = x**3
    return g

a   = -0.5                # First End Point
b   = 1.0               # Second End Point
TOL = 0.0001             # Accuracy Required

# N = 20


BisectionMethod(a,b,TOL)
~
~
```

```
*** BISECTION METHOD IMPLEMENTATION ***
iteration =   1    a =   -0.5000000000    b =    1.0000000000    m =    0.2500000000    f(m) =    0.0156250000
iteration =   2    a =   -0.5000000000    b =    0.2500000000    m =   -0.1250000000    f(m) =   -0.0019531250
iteration =   3    a =   -0.1250000000    b =    0.2500000000    m =    0.0625000000    f(m) =    0.0002441406
iteration =   4    a =   -0.1250000000    b =    0.0625000000    m =   -0.0312500000    f(m) =   -0.0000305176

Final Root Found

iteration =   4    a = -0.12500   b = 0.062500   m = -0.03125   f(m) = -0.00003
```

False Position Method

```
from math import *
MaxIterations = 10000


# Implementing False Position Method
def falsePosition(x0 ,x1 ,e ,N=MaxIterations):

    print('\n\n*** FALSE POSITION METHOD IMPLEMENTATION ***')

    step = 2

    while step <= N :

        # x2 = (x0*f(x1) - x1*f(x0))/( f(x1) - f(x0) )      # Both formula's are same
        x2 = x1 - (f(x1)*(x1-x0))/(f(x1) - f(x0))


        print(f'Iteration-{step:3}, x2 = {x2} and f(x2) = {f(x2)}')
        # print(f'Iteration = {step:3}, x1 = {x1}   Accuracy = {accuracy}   f(x1) = {f(x1)} ')

        if(abs(x2-x1)<e or f(x2) <e):
            print(f'\nRequired root is: {x2:3.06}')
            break

        if f(x0) * f(x2) < 0:
            x1 = x2
        else:
            x0 = x2

        step = step + 1

    if step > N:
        print('Not Convergent!')

# Defining Function
def f(x):

    g = cos(x) - x*exp(x)

    return g

# Input Section

x0 = 0.0
x1 = 1.0
e  = 0.0001

# N = 20

# x0 = float(input('First Guess: '))
# x1 = float(input('Second Guess: '))
# e = float(input('Tolerable Error: '))
# N = int(input('Number of Steps: '))


# Checking Correctness of initial guess values and false positioning
if f(x0) * f(x1) > 0.0:
    print('Given guess values do not bracket the root.')
    print('Try Again with different guess values.')
else:
    falsePosition(x0,x1,e)
~
```

```
*** FALSE POSITION METHOD IMPLEMENTATION ***
Iteration-  2, x2 = 0.314665337800771 and f(x2) = 0.5198711737709334
Iteration-  3, x2 = 0.4467281445913339 and f(x2) = 0.20354477776832092
Iteration-  4, x2 = 0.4940153365958987 and f(x2) = 0.07080234897836535
Iteration-  5, x2 = 0.509946140365247 and f(x2) = 0.023607718826251345
Iteration-  6, x2 = 0.5152010099022495 and f(x2) = 0.0077601137286161 8
Iteration-  7, x2 = 0.5169222100105166 and f(x2) = 0.0025388646887646305
Iteration-  8, x2 = 0.5174846767845119 and f(x2) = 0.0008293578901886756
Iteration-  9, x2 = 0.5176683449777302 and f(x2) = 0.0002707857277036707
Iteration- 10, x2 = 0.5177283052714122 and f(x2) = 8.839713027164464e-05

Required root is: 0.517728
```

Secant Method

```
from math import *

MaxIterations = 10000

# Implementing Secant Method
def secant(x0,x1,e,N=MaxIterations):
    print('\n\n*** SECANT METHOD IMPLEMENTATION ***')

    x0 = float(x0)
    x1 = float(x1)

    step = 2

    while step <= N:
        if f(x0) == f(x1):
            print('Divide by zero error!')
            break


        x2 = x1 - (x1-x0)*f(x1)/( f(x1) - f(x0) )

        if(abs(x2-x1)<e):
            print(f'\nRequired root is: {x1:3.06}')
            break

        print(f'Iteration-{step:3}, a = {x0}            b ={x1}           x{step} = {x2}             and f(x2) = {f(x2)}')
        # print(f'Iteration = {step:3}, x1 = {x1}   Accuracy = {accuracy}   f(x1) = {f(x1)} ')

        x0 = x1
        x1 = x2
        step = step + 1

    if(step > N):
        print('\nNot Convergent.')


# Defining Function
def f(x):
    return (x**3) - 10*(x**2) +5

# Input Section
x0 = 0.0
x1 = 1.0
e = 0.0001

# N = 20

#Note: You can take input from user like this
# x0 = float(input('Enter First Guess: '))
# x1 = float(input('Enter Second Guess: '))
# e = float(input('Tolerable Error: '))
# N = int(input('Maximum Step: '))

# Starting Secant Method
secant(x0,x1,e)
~
```

```
*** SECANT METHOD IMPLEMENTATION ***
Iteration-  2, a = 0.0           b =1.0          x2 = 0.5555555555555556             and f(x2) = 2.0850480109739364
Iteration-  3, a = 1.0           b =0.5555555555555556           x3 = 0.7078449053201081             and f(x2) = 0.3442176330725361
Iteration-  4, a = 0.5555555555555556            b =0.7078449053201081           x4 = 0.7379573621013965             and f(x2) = -0.04393307419850423
Iteration-  5, a = 0.7078449053201081            b =0.7379573621013965           x5 = 0.7345490653411698             and f(x2) = 0.0007117108135012984

Required root is: 0.734549
$
```

Newton's Method

```
from math import *

MaxIterations = 10000000

# Implementing Newton Raphson Method

def newtonRaphson(x0,e,N=MaxIterations):
    print('\n\n*** NEWTON RAPHSON METHOD IMPLEMENTATION ***')
    step = 1

    while step <= N:
        if g(x0) == 0.0:
            print('Divide by zero error!')
            break

        x1 = x0 - f(x0)/g(x0)

        print(f'Iteration = {step:3}, x1 = {x1:3}   f(x1) = {f(x1):3} ')
        # print(f'Iteration = {step:3}, x1 = {x1}   f(x1) = {f(x1)} ')

        if(abs(x1-x0)<e):
            print(f'\nRequired root is: {x1:3.06}')
            break
        x0 = x1
        step = step + 1

    if step > MaxIterations:
        print('\nNot Convergent.')



# Defining Function
def f(x):

    y = 3*cos(x-1) - x


    return y

# Defining derivative of function
def g(x):

    z = -3*sin(x-1) -1
    return z


# Input Section

x0 = 0
e  = 0.0001

# N  = 25

# x0 = float(input('Enter Guess: '))
# e = float(input('Tolerable Error: '))
# N = int(input('Maximum Step: '))

# Starting Newton Raphson Method
newtonRaphson(x0,e)
```

```
*** NEWTON RAPHSON METHOD IMPLEMENTATION ***
Iteration =   1, x1 = -1.0632990968101619   f(x1) = -0.35519896755507174
Iteration =   2, x1 = -0.8471699034071574   f(x1) = 0.028563968054829747
Iteration =   3, x1 = -0.862313931731673   f(x1) = 9.55390587460414e-05
Iteration =   4, x1 = -0.8623649286949181   f(x1) = 1.1212446526798203e-09

Required root is: -0.862365
$ vim 4-NewtonRaphsonMethod.py
```

Fixed Point Iteration Method

```
# Fixed Point Iteration Method
# Importing math to use sqrt function
from math import *

MaxIterations = 1000

# Implementing Fixed Point Iteration Method
def fixedPointIteration(x0, e, N = MaxIterations):
    print('\n\n*** FIXED POINT ITERATION METHOD ***')
    x0 = float(x0)
    step = 1

    while step <= N:
        x1 = g(x0)

        accuracy = abs(x1-x0)
        print(f'Iteration = {step:3}, x1 = {x1:3.15}   Accuracy = {accuracy:3.15}   f(x1) = {f(x1):3.15} ')

        # print(f'Iteration = {step:3}, x1 = {x1}   Accuracy = {accuracy}   f(x1) = {f(x1)} ')

        if accuracy < e:
            print(f'\nRequired root is: {x1:3.15}')
            break

        step = step + 1
        x0=x1

    if(step > N):
        print('\nNot Convergent.')

def f(x):
    return cos(x) - x*exp(x)

# Re-writing f(x)=0 to x = g(x)
def g(x):
    return (cos(x)/exp(x))

# Converting x0 and e to float
x0 = 0
e = 0.001

#Note: You can Take input from user using following commands
# x0 = float(input('Enter Guess: '))
# e = float(input('Tolerable Error: '))
# N = int(input('Maximum Step: '))

# Starting Newton Raphson Method
fixedPointIteration(x0,e)
```

```
*** FIXED POINT ITERATION METHOD ***
Iteration =   1, x1 = 1.0   Accuracy = 1.0   f(x1) = -2.17797952259091
Iteration =   2, x1 = 0.198766110346413   Accuracy = 0.801233889653587   f(x1) = 0.737836863283456
Iteration =   3, x1 = 0.803601681187326   Accuracy = 0.604835570840913   f(x1) = -1.10078295854652
Iteration =   4, x1 = 0.310766251200532   Accuracy = 0.492835429986793   f(x1) = 0.528068237231075
Iteration =   5, x1 = 0.697779629366032   Accuracy = 0.3870133781655   f(x1) = -0.635768409971049
Iteration =   6, x1 = 0.381364601116034   Accuracy = 0.316415028249998   f(x1) = 0.369732525627954
Iteration =   7, x1 = 0.633865608764166   Accuracy = 0.252501007648132   f(x1) = -0.389018267789133
Iteration =   8, x1 = 0.427477030409113   Accuracy = 0.206388578355053   f(x1) = 0.254528187095111
Iteration =   9, x1 = 0.59346819378143   Accuracy = 0.165991163372318   f(x1) = -0.245323157144793
Iteration =  10, x1 = 0.457949694591266   Accuracy = 0.135518499190165   f(x1) = 0.173020466605992
Iteration =  11, x1 = 0.567398859821614   Accuracy = 0.109449165230348   f(x1) = -0.157404559013158
Iteration =  12, x1 = 0.478150732303748   Accuracy = 0.0892481275178655   f(x1) = 0.116547858974919
Iteration =  13, x1 = 0.550402100442932   Accuracy = 0.0722513681391834   f(x1) = -0.102055496621341
Iteration =  14, x1 = 0.491544872295267   Accuracy = 0.058857228147665   f(x1) = 0.0780075846739973
Iteration =  15, x1 = 0.539260605928951   Accuracy = 0.047715733633684   f(x1) = -0.0666023404789087
Iteration =  16, x1 = 0.500419500142187   Accuracy = 0.0388411057867635   f(x1) = 0.0519829092854829
Iteration =  17, x1 = 0.531935504663029   Accuracy = 0.0315160045208414   f(x1) = -0.0436462748944867
Iteration =  18, x1 = 0.506294766179474   Accuracy = 0.0256407384835543   f(x1) = 0.0345373141161507
Iteration =  19, x1 = 0.52711125786332   Accuracy = 0.0208164916838455   f(x1) = -0.0286793104816722
Iteration =  20, x1 = 0.510181638451345   Accuracy = 0.0169296194119752   f(x1) = 0.0229004969297844
Iteration =  21, x1 = 0.523930788009228   Accuracy = 0.0137491495578831   f(x1) = -0.0188776094105908
Iteration =  22, x1 = 0.512751690458599   Accuracy = 0.0111790975506284   f(x1) = 0.0151641657004483
Iteration =  23, x1 = 0.521832682427854   Accuracy = 0.00908099196925438   f(x1) = -0.0124399891689685
Iteration =  24, x1 = 0.514450395043854   Accuracy = 0.00738228738399926   f(x1) = 0.0100323840687166
Iteration =  25, x1 = 0.520448045923324   Accuracy = 0.00599765087946924   f(x1) = -0.0082038468823542
Iteration =  26, x1 = 0.515572875169151   Accuracy = 0.00487517075417232   f(x1) = 0.00663333851039538
Iteration =  27, x1 = 0.519534028976637   Accuracy = 0.00396115380748618   f(x1) = -0.00541288097495607
Iteration =  28, x1 = 0.516314460136999   Accuracy = 0.0032195688396387   f(x1) = 0.00438419060973505
Iteration =  29, x1 = 0.518930575592695   Accuracy = 0.00261611545569618   f(x1) = -0.00357256323988697
Iteration =  30, x1 = 0.516804340705596   Accuracy = 0.00212623488709907   f(x1) = 0.00289690107610763
Iteration =  31, x1 = 0.518532120503489   Accuracy = 0.0017277797978934   f(x1) = -0.00235843545690795
Iteration =  32, x1 = 0.517127922481863   Accuracy = 0.00140419802162639   f(x1) = 0.00191382903458837
Iteration =  33, x1 = 0.518269005746019   Accuracy = 0.00114108326415596   f(x1) = -0.00155714536838913
Iteration =  34, x1 = 0.517341646964152   Accuracy = 0.00092735878186645   f(x1) = 0.00126422154157069

Required root is: 0.517341646964152
```

Modified Newton Method

```
MaxIterations = 1000

# Implementing Newton Raphson Method
def h(x):
    return f(x)/f1(x)
def g(x):
    return (((f1(x))**2)-(f(x))*f2(x))/((f1(x))**2)

def modifiedNewtonRaphson(x0,e,N=MaxIterations):
    print('\n\n*** NEWTON RAPHSON METHOD IMPLEMENTATION ***')
    step = 1

    while step <= N:
        if g(x0) == 0.0:
            print('Divide by zero error!')
            break

        x1 = x0 - h(x0)/g(x0)

        print(f'Iteration = {step:3}, x1 = {x1:3}   f(x1) = {f(x1):3} ')
        # print(f'Iteration = {step:3}, x1 = {x1}   f(x1) = {f(x1)} ')

        if(abs(x1-x0)<e):
            print(f'\nRequired root is: {x1:3.06}')
            break
        x0 = x1
        step = step + 1

    if step > MaxIterations:
        print('\nNot Convergent.')


# Defining Function
def f(x):
    return (x**3) + 4*(x**2) -10

# Defining derivative of function
def f1(x):
    return 3*(x**2) + 4*2*x

def f2(x):
    return 3*2*x + 8


# Input Section

x0 = 1.5
e  = 0.001
N  = 25

# x0 = float(input('Enter Guess: '))
# e = float(input('Tolerable Error: '))
# N = int(input('Maximum Step: '))

# Starting Newton Raphson Method
modifiedNewtonRaphson(x0,e,N)
~
~ ,
$ python3 6-ModifiedNewtonRaphsonMethodForQuadraticConvergence.py


*** NEWTON RAPHSON METHOD IMPLEMENTATION ***
Iteration =   1, x1 = 1.3568989756979313   f(x1) = -0.1370124377524533
Iteration =   2, x1 = 1.3651958490280898   f(x1) = -0.0005641606910256058
Iteration =   3, x1 = 1.3652300128418653   f(x1) = -9.449488302948339e-09

Required root is: 1.36523
```

```python
from math import *
import cmath
MAX_ITERATIONS = 10000


def Muller(x0, x1, x2, TOL, N = MAX_ITERATIONS):
    print("\n\n*** MULLER METHOD IMPLEMENTATION ***")
    h1 = x1 - x0
    h2 = x2 - x1

    f0 = f(x0)
    f1 = f(x1)
    f2 = f(x2)

    d1 = (f1 - f0)/h1
    d2 = (f2 - f1)/h2
    d = (d2-d1)/(h1+h2)
    i = 3

    while i<=N:

        b = d2 +h2*d
        D = ((b**2)-4*f2*d)**(1/2)

        if(abs(b-D) < abs(b+D)):
            E = b + D
        else:
            E = b - D

        h = -2*f2/E
        p = x2 + h

        if(abs(h)<TOL):
            print(f"\n\niteration = {i:3}      x = {p:f}        f(p) = {f(p):f}")
            break

        print(f'a = {d:f}               b = {b:f}              c = {f(x2):f}\n')
        print(f'\nIteration = {i:3}      p = {p:3.10f}          f(p) = {f(p):3.10f} ')
        x0 = x1
        x1 = x2
        x2 = p
        h1 = x1 - x0
        h2 = x2 - x1

        f0 = f(x0)
        f1 = f(x1)
        f2 = f(x2)

        d1 = (f1 - f0)/h1
        d2 = (f2 - f1)/h2
        d = (d2-d1)/(h1+h2)

        i += 1
    if (i > MAX_ITERATIONS):
        print("Root cannot be found using",
                        "Muller's method")




def f(x):

    g = (x**4) - 4*(x**2) - 3*x + 5
    # g = 16*(x**4) - 40*(x**3) +5*(x**2) + 20*x + 6
    return g


# Driver Code

# x0 = -1.5
# x1 = 2.0
# x2 = 2.1

x0 =complex(-1.5,0.0)
x1 =complex(-1.5,-1.0)
x2 =complex(-1.5,-0.9)

e = 10**(-5)

Muller(x0, x1, x2, e)
```

```
*** MULLER METHOD IMPLEMENTATION ***
a = 6.790000+11.400000j               b = 10.620000-13.932000j              c = -1.476400-0.324000j


Iteration =   3      p = -1.4605077013-0.8160286172j          f(p) = -0.0162820612-0.0920077259j
a = 4.338249+16.156683j               b = 7.773589-12.172469j               c = -0.016282-0.092008j


Iteration =   4      p = -1.4653023275-0.8116858031j          f(p) = -0.0005899891+0.0005558327j
a = 4.796956+14.918365j               b = 7.720113-12.284246j               c = -0.000590+0.000556j


Iteration =   5      p = -1.4652482502-0.8116717594j          f(p) = 0.0000000789+0.0000002173j


iteration =   6      x = -1.465248-0.811672j        f(p) = 0.000000-0.000000j
```