Questin 1(a)

```
import math
def f(x):
  return (2-math.exp(x)+x*x)/3
p1 = f(0.5)
p1
```

```
0.20042624309995727
```

```
p2 = f(0.20042624309995727)
p2
```

```
0.27274906509837465
```

```
p3 = f(0.27274906509837465)
p3
```

```
0.25360715658413
```

```
p4 = f(0.25360715658413)
p4
```

```
0.2585503762649362
```

```
p5 = f(0.2585503762649362)
p5
```

```
0.25726563633509353
```

```
p6 = f(0.25726563633509353)
p6
```

```
0.2575989851621903
```

```
p0 = 0.5
def p(x,y,z):
  return x-((y-x)*(y-x)/(x+z-2*y))
p_hat_0 = p(p0, p1, p2)
p_hat_0
```

```
0.2586844275657909
```

```
p_hat_1 = p(p1, p2, p3)
p_hat_1
```

```
0.2576132107157495
```

```
p_hat_2=p(p2,p3,p4)
p_hat_2
```

```
0.25753583232666766
```

```
p_hat_3 = p(p3,p4,p5)
p_hat_3
```

```
0.25753066000103225
```

```
p_hat_4 = p(p4,p5,p6)
p_hat_4
```

```
0.25753031065960186
```

Questin 1(b)

```
def f1(x):
  return math.pow(math.exp(x)/3, 0.5)
p_0 = 0.75
p_1 = f1(p_0)
p_1
```

```
0.840039684898413
```

```
p_2 = f1(p_1)
p_2
```

```
0.8787223496632842
```

```
p_3 = f1(p_2)
p_3
```

```
0.8958834348512263
```

```
p_4 = f1(p_3)
p_4
```

```
0.9036036753822647
```

```
p_5 = f1(p_4)
p_5
```

```
0.9070984349983345
```

```
p_6 = f1(p_5)
p_6
```

```
0.9086848661327573
```

```
p_hat0 = p(p_0,p_1,p_2)
p_hat0
```

```
0.9078585524534876
```

```
p_hat1 = p(p_1,p_2,p_3)
p_hat1
```

```
0.9095675068671811
```

```
p_hat2 = p(p_2,p_3,p_4)
p_hat2
```

```
0.9099168937459952
```

```
p_hat3=p(p_3,p_4,p_5)
p_hat3
```

```
0.9099888384860999
```

```
p_hat4 = p(p_4,p_5,p_6)
p_hat4
```

```
0.9100036976486471
```

Question 2

```
def newton( x, f, df, E ):
    h = f(x) / df(x)
    n=0
    while abs(h) >= E:
        h = f(x)/df(x)
        x = x - h
        n+=1
        print(n)
        print(x)
    print("The value of the root is : ",
                                 "%.10f"% x)
    print("The number of iterations is : ", "%d"%n)
```

```
def f2(x):
  return math.exp(6*x) + 3*math.log(2)*math.log(2)*math.exp(2*x) - math.log(8)
  *math.exp(4*x) - math.pow(math.log(2),3)

def df2(x):
  return 6*math.exp(6*x) + 6*math.log(2)*math.log(2)*math.exp(2*x) - 12
  *math.log(2)*math.exp(4*x)
```

```
    math.log(2, math.exp(-x)
```

```
newton(0,f2, df2, 0.0002)
```

```
1
-0.051142136573342496
2
-0.08984247581502683
3
-0.11824473602582486
4
-0.13856559584000824
5
-0.15281619296929916
6
-0.16266025259028719
7
-0.16938617562230754
8
-0.17394606447475225
9
-0.17702081377113918
10
-0.1790864552291608
11
-0.18047067668200603
12
-0.18139668915127077
13
-0.1820154613777542
14
-0.18242861474085212
15
-0.1827043349462318
16
-0.18288827513613412
The value of the root is :  -0.1828882751
The number of iterations is :  16
```

```
phat1 = p(0,-0.051142136573342496,-0.08984247581502683)
phat1
```

```
-0.21022028116719343
```

```
phat2 = p(-0.051142136573342496,-0.08984247581502683,-0.11824473602582486)
print(phat2)
print(abs(phat2-phat1))
```

```
-0.1965786061255237
0.013641675041669743
```

```
phat3 = p(-0.08984247581502683,-0.118244736025824864,-0.13856559584000824)
print(phat3)
print(abs(phat3-phat2))
```

```
    0.18066284673274504
```

```
phat4 = p(-0.118244736025824864,-0.13856559584000824,-0.15281619296929916)
print(phat4)
print(abs(phat4-phat3))
```

```
-0.18627100853218642
0.0033918381952639798
```

```
phat5 = p(-0.13856559584000824,-0.15281619296929916,-0.16266025259028719)
print(phat5)
print(abs(phat5-phat4))
```

```
-0.1846515574728567
0.0016194510593297295
```

```
phat6 = p(-0.15281619296929916,-0.16266025259028719,-0.16938617562230754)
print(phat6)
print(abs(phat6-phat5))
```

```
-0.18389421250328267
0.0007573449695740253
```

```
phat7 = p(-0.16266025259028719,-0.16938617562230754,-0.17394606447475225)
print(phat7)
print(abs(phat7-phat6))
```

```
-0.18354544501210024
0.0003487674911824279
```

```
phat8 = p(-0.16938617562230754,-0.17394606447475225,-0.17702081377113918)
print(phat8)
print(abs(phat8-phat7))
```

```
-0.18338660154984954
0.00015884346225070178
```

Section 3.1

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

# xi -> corresponds to the new data point
# n -> represents the number of known data points
def interpolate(f: list, xi: int, n: int) -> float:
    result = 0.0
    for i in range(n):
        term = f[i].y
        for j in range(n):
            if i != j:
```

```
            if j != i:
                term = term * (xi - f[j].x) / (f[i].x - f[j].x)
        result += term
    return result
```

Question 5a

```
# degree one
d1 = [Point(8.3, 17.56492), Point(8.6, 18.50515)]
interpolate(d1, 8.4, 2)
```

```
17.878330000000002
```

```
#degree two
d2 = [Point(8.3, 17.56492), Point(8.6, 18.50515), Point(8.7, 18.82091)]
interpolate(d2, 8.4, 3)
```

```
17.877155000000002
```

```
#degree three
d3 = [Point(8.1, 16.94410), Point(8.3, 17.56492), Point(8.6, 18.50515), Point(8.7,
interpolate(d3, 8.4, 4)
```

```
17.877142500000005
```

Question 5b

```
# degree one
d4 = [Point(-0.5, -0.02475), Point(-0.25, 0.3349375)]
interpolate(d4, -1/3, 2)
```

```
0.2150416666666667
```

```
#degree two
d5 = [Point(-0.5, -0.02475), Point(-0.25, 0.3349375), Point(0, 1.101)]
interpolate(d5, -1/3, 3)
```

```
0.1698888888888889
```

```
#degree three
d6 = [Point(-0.5, -0.02475), Point(-0.25, 0.3349375), Point(0, 1.101),
      Point(-0.75, -0.0718125)]
interpolate(d6, -1/3, 4)
```

```
0.17451851851851855
```

Question 15a

```
d7 = [Point(0, 1), Point(0.5, math.exp(0.5))]
interpolate(d7, 0.25, 2)
```

```
1.324360635350064
```

```
d8 = [Point(0.5, math.exp(0.5)), Point(1, math.exp(1))]
interpolate(d8, 0.75, 2)
```

```
2.183501549579587
```

```
d9 = [Point(0, 1), Point(1, math.exp(1)), Point(2, math.exp(2))]
interpolate(d9, 0.25, 3)
```

```
1.1527742906760838
```

```
interpolate(d9, 0.75, 3)
```

```
2.0119152049056064
```