```
from sympy import diff, sympify
from sympy.abc import x

class NewtonRaphsonLak:
  MAX_ITER = 100
  TOLERANCE = 0.001

  def __init__(self, expression = None, initial = 0, number = -1):
    self.expression = expression
    self.initial = initial
    self.number = number

  def set_expression(self, expression):
    self.expression = expression

  def set_inital_value(self, initial):
    self.initial = initial

  def set_number(self, number):
    self.number = number

  def find_equation_root(self):
    # Assign the initial value of x to find the tanget at that point in the curve.
    x_begin = self.initial
    flag = False
    print('\nx_0 = ', x_begin)

    for iteration_no in range(self.MAX_ITER):
      try:
        # Find the diiereniation of the equation
        differentiation = diff(self.expression, x)
        # Find the next possible x value where the tanget at x_begin intersects the x-axis
        x_next = x_begin - float(self.expression.subs(x, x_begin)) / float(differentiation.subs(x, x_begin))
        print('x_{iter_no} = '.format(iter_no = iteration_no + 1), x_next)

      # Catch ZeroDivisionError
      except ZeroDivisionError  as div_zero_err:
        print('ZeroDivisionError: ', div_zero_err)
        print('The drivative of the expression entered cannot be zero!')
      else:
        # Check for closeness
        if abs(x_next - x_begin) < self.TOLERANCE:
          flag = True
          print('\nApproximate root found is: ', x_next)
          break
        else:
          # Update x_begin
          x_begin = x_next

    if flag == False:
      print('\nThe given expression does not converge to a root within the specified number of iterations')

  def find_square_root(self):
    num = self.number
    try:
      # Check using Assert statement
      assert num > 0, 'The number entered must be a postive number!'

    # Catch AssertionError
    except AssertionError as assert_err:
      print('AssertionError: ', assert_err)
    else:
      # Assuming the sqrt of num as num itself
      assumed_sqrt = num
      while 1:
        try:
          sq_root = 0.5 * (assumed_sqrt + (num / assumed_sqrt))

        # Catch ZeroDivisionError
        except ZeroDivisionError  as div_zero_err:
          print('ZeroDivisionError: ', div_zero_err)
          print('The number cannot be zero!')
          break
        else:
          # Check for closeness
          if (abs(sq_root - assumed_sqrt) < self.TOLERANCE):
            print('\nApproximate square root found is: ', sq_root)
            break

          # Update root
          assumed_sqrt = sq_root
```

```python
  def main(self):
    print('To find the root of an equation by Newton-Raphson method:\n')
    print('Enter the equation: (Eg: 4*x^3 + x + 2)')
    equation = sympify(input())

    print('Enter the initial guess value of the root of the equation: ')
    init_val = int(input())

    self.set_expression(equation)
    self.set_inital_value(init_val)
    self.find_equation_root()

    print('-----------------------------------------------------------------------------')

    print('\n\nTo find the square root of a positive number using Newton-Raphson method:\n')
    print('Enter the number:')
    number = int(input())

    self.set_number(number)
    self.find_square_root()





if __name__ == "__main__":
  NewtonRaphsonLak().main()
```

```
To find the root of an equation by Newton-Raphson method:

Enter the equation: (Eg: 4*x^3 + x + 2)
4*x^3 + x + 2
Enter the initial guess value of the root of the equation:
3

x_0 =  3
x_1 =  1.963302752293578
x_2 =  1.2388464923727205
x_3 =  0.6803595237137808
x_4 =  0.07924854206258025
x_5 =  -1.8561327938903083
x_6 =  -1.2554315144121815
x_7 =  -0.895359815126686
x_8 =  -0.7290241933720747
x_9 =  -0.6912265240980129
x_10 =  -0.6894024635591963
x_11 =  -0.6893983500856581

Approximate root found is:  -0.6893983500856581
-----------------------------------------------------------------------------


To find the square root of a positive number using Newton-Raphson method:

Enter the number:
16

Approximate square root found is:  4.000000000000051
```