# 云南大学数学与统计学院
# 上机实践报告

Code Listing 1: **ff.py**

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Dec 24 21:28:46 2017

@author: Newton
"""
"""filename: get_root.py"""

import math

class root:
    """This class provides some ways to find roots"""

    def __init__(self, fun_name, x_left, x_right = None):
        """fun_name represents the name of the function if the equation.

        Both left and right ends will be given by x_left and x_right.
        """
        if x_right != None:
            """Only support binary method."""

            if fun(x_left) * fun(x_right) < 0:
                self.x_l = x_left
                self.x_r = x_right
                self.fun = fun_name
                self.method = 'binary'
            else:
                raise ValueError("values on x_right and x_left should have opposite sign.")
        else:
            """Only support Aitken method."""
            self.x = x_left
            self.fun_after_convert = fun_name
            self.method = 'Aitken'

    def binary(self, e):
        """e = (b - a) / 2"""
        if self.method != 'binary':
            raise ValueError("Method does not support!")
        a = self.x_l
        b = self.x_r
        times = 0

        while ( abs(b-a)/2) > e:
            if self.fun((a + b)/2) == 0:
                return (a + b)/2
```

```
46              elif self.fun(a) * self.fun((a + b)/2) < 0:
47                  b = (a + b)/2
48              else:
49                  a = (a + b)/2
50              times += 1
51          ans = ((a + b)/2, times)
52          return ans
53
54      def aitken(self, e):
55          """e = x - x0"""
56          if self.method != 'Aitken':
57              raise ValueError("Method does not support!")
58          x0 = self.x
59          x1 = self.fun_after_convert(x0)
60          x2 = self.fun_after_convert(x1)
61
62          x = x0 - (x1-x0)*(x1-x0)/(x2-2*x1+x0)
63          times = 1
64
65          while  abs(x-x0) > e:
66              x0 = x
67              x1 = self.fun_after_convert(x0)
68              x2 = self.fun_after_convert(x1)
69
70              x = x0 - (x1-x0)*(x1-x0)/(x2-2*x1+x0)
71
72              times += 1
73          ans = (x, times)
74          return ans
75
76  if __name__ == "__main__":
77      def fun(x):
78          return x - math.tan(x)
79
80      def fun_after_convert(x):
81          return math.tan(x)
82
83      c = root(fun, 4.4, 4.6)
84      e = 0.00001
85      tmp = c.binary(e)
86      print('root.binary(x - tan(x) == 0) is ', tmp[0], 'iterations deepth:', tmp[1])
87      d = root(fun_after_convert, 4.5)
88      tmp = d.aitken(e)
89      print('root.aitken(x - tan(x) == 0) is ', tmp[0], 'iterations deepth:', tmp[1])
```