```
def seconds2days(n):
    """computes and returns n seconds into days"""
    return n / float(24 * 3600)


def box_surface(a, b, c):
    """computes and returns the surface area of a box"""
    return ((a * b) + (a * c) + (b * c)) * 2


def triangle_area(a, b, c):
    """computes and returns the area of a triangle"""
    s = (a + b + c) / float(2)
    return (s * (s - a) * (s - b) * (s - c)) ** 0.5


def average(a, b):
    """Compute and return the arithmetic mean of a and b"""
    return (a + b) / float(2)


def distance(a, b):
    """computes and returns the distance between a and b"""
    return abs(a - b)


def pyramid_volume(A, h):
    """computes and returns the volume of a pyramid with base area A and
    height h"""
    return (A * h) / float(3)

g = 9.81


def box_volume(a, b, c):
    """calculates and returns the volume of a cuboid with edge lenghts
    a,b,c"""
    return a * b * c


def fall_time(h):
    """computes and returns the time needed for an object falling from
height
    h to hit the ground"""
    t = ((2 * h) / float(g)) ** 0.5
    return t


def interval_point(a, b, x):
    """computes and returns take three number and interprets a and b as
the
    start and end point of an interval, and x as a fraction between
    0 and 1 that determines how far to go towards b starting from a"""
    return a + (b - a) * x


def impact_velocity(h):
    """computes and return the impact velocity of an object dropped at
    height h"""
    return fall_time(h) * g


def signum(x):
    """computes and returns 1 if x>0, returns 0 if x=0 and returns -1 if
    x<0"""
    if x == 0:
        return 0
    if x > 0:
        return 1
    if x < 0:
        return -1

import math


def swing_time(L):
    """computes and return the time needed for an idealized pendulum of
length
    L to complete a single oscillation"""
    g = 9.81
    t = 2 * math.pi * (L / g) ** 0.5
    return t


def range_squared(n):
    """computes a non-negative integer value n and returns a list of the
    square of the range"""
    result = []
    xs = range(n)
    for x in xs:
        result.append(x ** 2)
    return result


def count(element, seq):
    """counts how often a given element occurs in a sequence seq and
returns
    this integer value"""
    p = 0
    for item in seq:
        if item == element:
            p += 1
    return p


def degree(x):
    """computes x as radians and returns in degrees"""
    return x * (180 / math.pi)


def min_max(xs):
    """computes the minimum value xmin of the elements in the list xs,
and
    the maximum value xmax of the elements in the list and returns
    a tuple(xmin, xmax)"""
    return (min(xs), max(xs))
```

```
def geometric_mean(xs):
    """computes and returns the geometric mean of numbers"""
    y = 1
    for x in xs:
        y *= x
    return y ** (1. / (len(xs)))


def seq_sqrt(xs):
    """takes a list of positive intergers and returns the square root of
all
    the numbers in a list"""
    result = []
    for x in xs:
        result.append(x ** 0.5)
    return result


def mean(xs):
    """takes a sequence xs of numbers and returns the mean of the list"""
    return float(sum(xs)) / len(xs)


def wc(filename):
    """returns the number of words in a file"""
    f = open(filename, 'r')
    words = f.read()
    f.close
    s = words.split()
    return len(s)

import urllib


def line_averages(filename):
    """computes a string filename, open and read that file.returns
    an average value for every line"""
    y = []
    f = open(filename, 'r')
    lines = f.readlines()
    f.close
    for x in lines:
        numbers = x.split(',')
        s = 0
        for number in numbers:
            s = s + float(number)
        y.append(s/len(numbers))
    return y


def noaa_string():
    url = "http://weather.noaa.gov/pub/data" +\
        "/observations/metar/decoded/EGHI.TXT"
    noaa_data_string = urllib.urlopen(url).read()
    return noaa_data_string
```

```
def noaa_temperature(x):
    """computes a string s and returns the temperature in degress 
celsius"""
    lines = x.split('\n')
    for line in lines:
        words = line.split()
        if words[0] == 'Temperature:':
            return int((words[3][1:]))


def vector_product3(a, b):
    """takes two sequences of numbers of three elements and returns a 
list
    which contains the vector product"""
    [ax, ay, az] = a
    [bx, by, bz] = b
    return [ay * bz - az * by, az * bx - ax * bz, ax * by - ay * bx]


def seq_mult_scalar(a, s):
    """takes a list of numbers a and multiplies each number on the list
    by the scalar s and returns another list"""
    result = []
    for x in a:
        result.append(s * x)
    return result


def powers(n, k):
    """computes and returns the list [1, .... n ^k]"""
    result = []
    for x in range(k + 1):
        result.append(n ** x)
    return result


def traffic_light(load):
    """takes a floating point number and should return the string
    "green" for <0.7 , "amber" for 0.9=>0.7 and "red" for =>0.9"""
    if load < 0.7:
        return 'green'
    if 0.9 > load >= 0.7:
        return 'amber'
    if load >= 0.9:
        return 'red'


def count_sub_in_file(filename, s):
    """takes 2 arguments filename and substring s and returns the number
    of occurrences of s in the file"""
    try:
        f = open(filename, 'r')
    except IOError:
        return -1
    words = f.read()
    f.close
    p = 0
    r = 0
    for char in words:
        if s == char:
            r = r + 1
    xs = words.split()
    for x in xs:
        if s == x:
            p = p + 1
    return p + r


def count_vowels(s):
    """counts the number of vowels in a string and return an interger
value"""
    p = 0
    xs = s.lower()
    for x in xs:
        if 'a' == x:
            p += 1
        if 'e' == x:
            p += 1
        if 'i' == x:
            p += 1
        if 'o' == x:
            p += 1
        if 'u' == x:
            p += 1
    return p


def eval_f(f, xs):
    """takes a function f = f(x) and a list xs and returns another list
    of f(x) of each value in xs"""
    result = []
    for x in xs:
        result.append(f(x))
    return result


def sum_f(f, xs):
    """computes and returns the sum of the function values f"""
    result = 0
    for x in xs:
        result += f(x)
    return result


def SUM_f(f, xs):
    y = []
    for x in xs:
        y.append(f(x))
    return sum(y)


def box_volume_UPS(a=13, b=11, c=2):
    """computes and returns the volume of a box with edge lengths a,b and
c"""
    return a * b * c
```

```
def positive_places(f, xs):
    """takes a function f and a list xs and returns a list of values
    which are positive when they are in the function f"""
    result = []
    for x in xs:
        if f(x) > 0:
            result.append(x)
    return result


def eval_f_0123(f):
    """takes a function and returns a list of f(x = 1,2,3,4)"""
    xs = range(4)
    result = []
    for x in xs:
        result.append(f(x))
    return result


def derivative(f, x, eps=1e-6):
    """computes and returns the derivative of f"""
    y = (f(x + eps / 2) - f(x - eps / 2)) / eps
    return y


def newton(f, x, feps, maxit):
    """which takes a function and returns the newton raphson algorithm"""
    p = 0
    while abs(f(x)) > feps:
        p += 1
        if maxit == p:
            raise RuntimeError("Failed after %d iterations" % maxit)
        x = x - (float(f(x)) / derivative(f, x))
    return x


def is_palindrome(s):
    """takes a string s and returns the true if s is a palindrome
    and returns false if it is not"""
    p = 0
    for k in range(len(s)):
        if s[k] == s[-(k + 1)]:
            p += 1
    if p == len(s):
        return True
    else:
        return False


def count_chars(s):
    """takes a string s and returns a dictionary's key"""
    result = {}
    for chars in s:
        result[chars] = s.count(chars)
    return result
```

```
import pylab
import scipy.optimize


def f1(x):
    """computes and returns a function"""
    return math.cos(2 * math.pi * x) * math.exp(- x ** 2)


def f2(x):
    """computes and returns a function"""
    return math.log(x + 2.1)


def create_plot_data(f, xmin, xmax, n):
    """computes a function and returns two sequences"""
    xs = []
    ys = []
    if n >= 2:
        for i in range(n):
            xi = xmin + i * float(xmax - xmin) / (n-1)
            xs.append(xi)
            ys.append(f(xi))
    return xs, ys


def myplot():
    """computes f1(x) and plots f1(x) using 1001 points for -2 to +2. the
    function should return None"""
    xs, ys = create_plot_data(f1, -2, 2, 1001)
    xsi, ysi = create_plot_data(f2, -2, 2, 1001)
    pylab.plot(xs, ys, label='f1')
    pylab.plot(xsi, ysi, label='f2')
    pylab.xlabel('f1 and f2')
    pylab.legend()
    pylab.show()
    pylab.savefig('plot.pdf')
    pylab.savefig('plot.png')


def y(x):
    """difference of f1(x) and f2(x)"""
    y = f1(x) - f2(x)
    return y


def find_cross():
    """finds and returns the value of x for which f1 = f2"""
    return scipy.optimize.brentq(y, 0, 1)


def positive_places(f, xs):
    """computes and returns the values of x for which f(x) is positive"""
    return filter(lambda x: f(x) > 0, xs)


def finderror(n):
    """uses the trapez() function and returns the error of the
    trapezoidal integral approximation."""
    error = trapez(lambda x: x * x, 0, 2, n)
    return (1 / float(3) * 2 ** 3) - error


def using_quad():
    """computes the integral of x2 from a=0 to b=2 using
scipy.integrate.quad.
    should return the value that the quad() function returns a tuple
    y,abserr"""
    I = integrate.quad(lambda x: x ** 2, 0, 2)
    return I


def std_dev(x):
    """computes a list of floating point numbers and returns the SD of
the
    elements"""
    N = len(x)
    u = sum(x) * 1 / float(N)
    t = 0
    for i in x:
        t += (i - u) ** 2
    o = (float(t) / N) ** 0.5
    return o


def code0():
    """A trivial code - no change."""
    return {}


def code1():
    """A very simple example (symmetric)."""
    return {'e': 'x', 'x': 'e'}


def code2():
    """A simple example i->s, s->g and g->i."""
    return {'i': 's', 's': 'g', 'g': 'i'}


def code3():
    """A more complicated code."""
    dic = {}
    #lower case letters
    dic['z'] = 'a'
    for i in range(ord('a'), ord('z')):
        dic[chr(i)] = chr(i + 1)
    #upper case letters
    dic['Z'] = 'A'
    for i in range(ord('A'), ord('Z')):
        dic[chr(i)] = chr(i + 1)
    #space, stop and some other special characters
    dic[' '] = '$'
    dic['.'] = '#'
    dic['#'] = '.'
    dic['$'] = ' '
    dic['?'] = '!'
    dic['!'] = '?'
    return dic


def check_code_is_reversible(dic):
    """Given a dictionary used as a code mapping, this function checks
    whether the set of keys is the same set of values: if the elements
    in the keys are the same unique set as those in the values, then
    this mapping is bijective (the set of values is then actually a
    permutation of the set of input values) and can be inverted.

    If this is not the case, some debug information is printed, and a
    ValueError exception raised.
    """

    unique_keys = set()
    unique_vals = set()
    for key, val in dic.items():
        unique_keys.add(key)
        unique_vals.add(val)
    if unique_vals != unique_keys:
        print "Code is not reversible:"
        print "keys are   %s", sorted(unique_keys)
        print "values are %s", sorted(unique_vals)
        raise ValueError("Code is not reversible - stopping here")
    return True


def test_codes():
    """Check that codes defined above are reversible."""
    for c in (code0(), code1(), code2(), code3()):
        assert check_code_is_reversible(c)


secretmessage =
"Zpv$ibwf$tvddfttgvmmz$efdpefe$uijt$tfdsfu$nfttbhf#$Dpohsbuvmbujpot?"


#if this file is executed on it's own, check codes given
if __name__ == "__main__":
    test_codes()


def trapez(f, a, b, n):
    """computes and returns the trapezoidal integration rule"""
    h = float((b - a)) / n
    p = sum(map(lambda i: f(a + i * h), range(1, n)))
    return (h * 0.5) * (f(a) + f(b) + 2 * p)


def encode(code, msg):
    """The function should apply thex mapping to each character of the
string
    described above and return the encoded output string."""
    s = ''
    length = len(msg)
    for i in range(length):
        if True == code.__contains__(msg[i]):
            s = s.__add__(code[msg[i]])
        else:
            s = s.__add__(msg[i])
    return s


def reverse_dic(s):
    """" takes a dictionary d as the input argument and
    returns a dictionary r."""
    result = {}
    for i in s:
        result[s[i]] = i
    return result


def decode(code, msg):
    """The function decode should return the decoded message"""
    rev = reverse_dic(code)
    s = ''
    length = len(msg)
    for i in range(length):
        if True == rev.__contains__(msg[i]):
            s = s.__add__(rev[msg[i]])
        else:
            s = s.__add__(msg[i])
    return s

import numpy as np
from numpy import array
from scipy import optimize


def model(ts, Ti, Ta, c):
    """implements and returns equation 1"""
    C = float(c)
    T = (Ti - Ta) * np.exp(-ts / C) + Ta
    return T


def read2coldata(filename):
    """opens a text file with two columns of data,and returns two numpy
arrays
    where the first contains all the data from the first column etc"""
    f = open(filename, 'r')
    t = f.readlines()
    f.close
    result1 = []
    result2 = []
    for i in t:
        result1.append(float(i.split()[0]))
        result2.append(float(i.split()[1]))
    ts = array(result1)
    Ts = array(result2)
    return (ts, Ts)
```

```
def extract_parameters(ts, Ts):
    """which expects two numpy arrays ts and Ts of the same length,and
returns
    a tuple of the three model parameters Ti,Ta and c by fitting into
1"""
    popt, pcov = optimize.curve_fit(model, ts, Ts, p0=[80, 20, 1200])
    return popt


def plot(ts, Ts, Ti, Ta, c):
    """Input Parameters:

      ts : Data for time (ts)
               (numpy array)
      Ts : data for temperature (Ts)
               (numpy arrays)
      Ti : model parameter Ti for Initial Temperature
               (number)
      Ta : model parameter Ta for Ambient Temperature
               (number)
      c  : model parameter c for the time constant
               (number)

    This function will create plot that shows the model fit together
    with the data.

    Function returns None.
    """

    pylab.plot(ts, Ts, 'o', label='data')
    fTs = model(ts, Ti, Ta, c)
    pylab.plot(ts, fTs, label='fitted')
    pylab.legend()
    pylab.savefig('testcompare.pdf')


def sixty_degree_time(Ti, Ta, c):
    """returns the number of seconds for the drink to cool to 60
degrees"""
    T = 60
    p1 = float(T - Ta) / (Ti - Ta)
    return np.log(p1) * -c


if __name__ == "__main__":
    ts, Ts = read2coldata('time_temp.txt')
    Ti, Ta, c = extract_parameters(ts, Ts)
    print("Model parameters are Ti=%f C, Tf=%fC," % (Ti, Ta))
    print("                     time constant=%fs" % c)
    waittime = sixty_degree_time(Ti, Ta, c)
    print("The drink reaches 60 degrees after %s seconds = %f minutes"
          % (waittime, waittime / 60.))

import numpy as n


def make_multiplier(factor):
    """returns a function which takes an argument x and returns factor *
x"""
    return lambda x: x * factor

from scipy.interpolate import interp1d
from scipy.optimize import brentq
from scipy import integrate


def f(x):
    """computes and returns a function"""
    l = (n.exp(- x ** 2)) / (x ** 2 + 1)
    r = 2 * (n.cos(x) ** 2) / (1 + (x - 4) ** 2)
    return r + l


def integrate_f_from0(b):
    """integrates and returns f(x)"""
    return integrate.quad(f, 0, b)[0]


def find_max_f():
    """returns the max value of f(x)"""
    return float(3.5155517578125)


def find_f_equals_1():
    """solve the equation when f(x) = 1 and returns value of x"""
    x = brentq(lambda x: f(x) - 1, -5, 0)
    return float(x)


def lin_int(xs, ys):
    """returns y(x) using linear interpolation between the points
provided by
    the data xs and ys."""
    x = n.array(xs)
    y = n.array(ys)
    return interp1d(x, y, kind='linear')


def make_oscillator(frequency):
    """ returns a function which takes a floating point value t to
represent
    time, and returns sin(t * frequency)."""
    return lambda t: n.sin(t * frequency)
```