```
"""     Superposition Eye Pathlength and Absorption Program
        Original QBASIC version by Magnus L Johnson and Genevre Parker, 1995
        Python rewrite by Stephen P Moss, 2012-2013
        http://about.me/gawbul
        gawbul@gmail.com
"""

__author__ = "Steve Moss"
__copyright__ = "Copyright 1995-2013, Magnus L Johnson and Stephen P Moss"
__credits__ = ["Steve Moss", "Magnus Johnson", "Genevre Parker"]
__license__ = "GPLv3"
__version__ = "0.42b"
__maintainer__ = "Steve Moss"
__email__ = "gawbul@gmail.com"
__status__ = "beta"

# import modules
import os, sys, time, re # needed for os, system, time and regular expression specific functions
from datetime import timedelta, date # needed for time specific functions
import math # needed for math functions (self.pi, cos, sin, tan, atan)
import getopt # needed to get options from command line
import rpy2 # needed for plotting subroutines in R
#import pygame # needed for graphics output *** not yet implemented ***

# main handler subroutine
def main():
        """Controls the main program flow."""
        # check what the program arguments are and assign appropriate variables
        opts_array = handle_options(sys.argv[1:])
        (input_file, graphicsopt) = opts_array

        # check whether the user provide an input filename
        if input_file:
                # process file
                process_input_file(input_file, graphicsopt)
                sys.exit()
        else:
                # just continue with inline parameters below
                pass

        # show startup information
        startup()

        # track how long it takes
        start = time.time()

        # if not using an input file for the parameters you can set them manually as follows
        # setup nephrops_eye as new SuperpositionEye object - with relevant parameters passed
        # using Nephrops norvegicus flat lateral measurments
        # see README file or GitHub for information on parameters
        print "Setting up new superposition eye object..."
        nephrops_eye = SuperpositionEye("nephrops", 180, 25, 7800, 50, 3200, 1.34, 1.37, 18, 0)

        # run the model
        print "Running the ray tracing model (please wait)..."
        nephrops_eye.run_model(graphicsopt)

        # summarise the data
        print "Outputting summary data..."
        nephrops_eye.summarise_data()

        # how long did we take?
```

```
        end = time.time()
        took = end - start
        print "\nFinished in %s seconds.\n" % timedelta(seconds=took)

# handle any program input options given at the command line
def handle_options(optsargs):
        """Handles the input arguments to the program."""
        # process using getopts
        try:
                (opts, args) = getopt.getopt(optsargs, "f:gchv", ["file=", "grap
, "citation", "help", "version"])
        except getopt.GetoptError as err:
                print str(err)
                usage()
                sys.exit(2)
        filename = None
        graphicsopt = False
        for o, a in opts:
                if o in ("-f", "--file"):
                        filename = a
                elif o in ("-g", "--graphics"):
                        graphicsopt = True
                        sys.exit()
                elif o in ("-c", "--citation"):
                        startup()
                        sys.exit()
                elif o in ("-h", "--help"):
                        usage()
                        sys.exit()
                elif o in ("-v", "--version"):
                        version = __version__
                        print "pathlen.py version %s" % version
                        sys.exit()
                else:
                        assert False, "unhandled option"
        return filename, graphicsopt

# display startup information in the terminal
def startup():
        """Displays information about the program on startup, or via the citation input argument."""
        print "\nPathLength - Implements a ray tracing model to calculate resolution and sensitivity in r
e superposition compound eyes."
        print "-" * len("PathLength - Implements a ray tracing model to calculate resolution and se
y in reflective superposition compound eyes.")
        print "If you use this program, please cite:"
        print "\nGaten, E., Moss, S., Johnson, M. 2013. The Reniform Reflecting Superposition Compou
es of Nephrops Norvegicus: Optics, \n" \
        "Susceptibility to Light-Induced Damage, Electrophysiology and a Ray Tracing Model. In: M. L. Jo
and M. P. Johnson, ed(s).\n" \
        "Advances in Marine Biology: The Ecology and Biology of Nephrops norvegicus. Oxford: Academ
s, 107:148."
        print "-" * len("Susceptibility to Light-Induced Damage, Electrophysiology and a Ray Tra
odel. In: M. L. Johnson and M. P. Johnson, ed(s).") + "\n"

        return

# display usage information to the terminal
def usage():
        """Displays usage information via the help input argument."""
        print "The valid program options are:"
        print "\t-f or --file\t\tAllows the user to provide a csv input file with sets\n\t\t\t\tof parameters f
vidual runs on individual lines."
        print "\t-g or --graphics\tTurn graphics on or off. *** not yet implemented ***"
        print "\t-c or --citation\tDisplays the citation information."
        print "\t-h or --help\t\tDisplays this usage information."
        print "\t-v or --version\t\tDisplays the program version."
        print "\n\tFor more information visit https://github.com/gawbul/pathlength/\n\tor email Steve Moss (gawbul@gmail.com)."
        return

# process the parameter input file
def process_input_file(filename, graphicsflag):
        """Processes a csv input file for multiple parameter model testing."""
        # first check the file exists and exist with error message if not
        if not os.path.exists(filename):
                print "Error: Filename \'%s\' does not exist" % filename
                sys.exit()

        # track how long it takes
        start = time.time()

        # file must exist, so open and parse
        inputfile = open(filename, "r")
        count = 1
        for line in inputfile.readlines():
                # check if it is a comment line

                # filter out whitespace and any dodgy characters and split into parts
                parts = re.sub('\s\W', '', line).split(",")

                # check we have the right number of parameters
                if len(parts) == 0:
                        # this just means that there was a blank line?
                        continue
                if len(parts) != 10:
                        print "Error: The number of parameters is incorrect on line %d." % count
                        continue

                # assign the variables from the parts list
                (sn, rl, rw, ed, fw, ad, cri, rri, bce, pra) = parts

                # create an object
                print "Setting up new superposition eye object..."
                eye_object_from_file = SuperpositionEye(str(sn), int(rl), float(rw), int(ed), float(fw), int(ad), float(cri), float(rri), int(bce), float(pra))

                # run the model
                print "Running the ray tracing model (please wait)..."
                eye_object_from_file.run_model(graphicsflag)

                # summarise the data
                print "Outputting summary data..."
                eye_object_from_file.summarise_data()

                # increment line count
                count += 1

        # how long did we take?
        end = time.time()
        took = end - start
        print "\nFinished in %s seconds.\n" % timedelta(seconds=took)
        return

# setup superposition eye class
class SuperpositionEye():
        def __init__(self, sn, rl, rw, ed, fw, ad, cri, rri, bce, pra):
                """Initialises the default variables of a new SuperpositionEye object."""
                # store parameters incase needed in future
                self.eye_parameters = [sn, rl, rw, ed, fw, ad, cri, rri, bce, pra]

```

```
                # set variables for output data
                self.rowdata = []
                self.output_data = []
                self.aa = 100*[0]
                self.ab = 100*[0]

                # set variables for calculations
                self.pi = math.pi # define self.pi
                self.conv = self.pi / 180 # convert radians to degrees (1 d
 = self.pi / 180 radians)
                self.proximal_rhabdom_angle = pra # used for pointy rhabdom

                # set files for output data
                self.setup_files(sn) # passes species name to prepend outpu
enames

                self.iteration_count = 1 # q = 0 in original
                self.shielding_pigment_length = 0.0 # extent of shielding p
t set to zero

                # check the blur circle extent isn't set to less than 1 oth
e we will get division by zero error
                if bce < 1:
                        bce = 1
                self.blur_circle_extent = bce # blur circle extent

                # input data - eye parameters
                self.rhabdom_length = float(rl) # rhabdom length
                self.increment_amount = self.rhabdom_length / 10 # amount t
rement tapetum or pigment
                self.reflective_tapetum_length = 0.0 # extent of tapetal pi
 set to zero

                self.num_facets = 0 # num of facets across aperture
                self.rhabdom_width = rw # rhabdom width/diameter
                self.aperture_diameter = ad # aperture diameter
                self.y = 0 # y??? - set to one originally, but we use 0 bas
dexing in python
                self.facet_width = fw # facet width
                self.eye_diameter = ed # eye diameter

                # undeclared in original code
                self.boa = 0 # boa???
                self.tot = 0 # tot???
                self.col_total = 0 # total rhaboms?
                self.row_total = 0 # total facets?

        def initial_calculations(self):
                """Does some initial calculations before running the main model."""
                # do initial calculations
                (sn, rl, rw, ed, fw, ad, cri, rri, bce, pra) = self.eye_par
rs      # get stored parameters

                self.eye_circumference = self.pi * self.eye_diameter # circ
ence of eye
                self.aperture_radius = self.aperture_diameter / 2 # aa in o
al code - aperture radius
                self.eye_radius = self.eye_diameter / 2 # eye radius
                self.da = math.sqrt((self.eye_radius ** 2) - (self.aperture
us ** 2)) # DA???
                self.ac = math.atan(self.aperture_radius / self.da) / self.
# AC???
                self.aperture_diameter = (self.ac / 360) * self.eye_circumf
e # change aperture diameter
                self.optical_axis = (self.facet_width / self.eye_circumfere
* 360 # calculate optical axis from eye circumference and facet width
```

```

        self.facet_num = 1 # facet number
        self.num_facets = int(self.aperture_diameter / self.facet_width) # num of facets across aperture
        self.rhabdom_radius = self.rhabdom_width / 2 # rhabdom radius
        self.old_rhabdom_length = self.rhabdom_length # old rhabdom length
        self.max_rhabdom_length = self.rhabdom_length # store rhabdom length for main loop
        self.inter_ommatidial_angle = 0 # inter-ommatidial angle
        self.current_facet = 0 # current facet

        # angle of total internal reflection (rhabdoms)
        self.cytoplasm_ri = cri # cytoplasm refractive index
        self.rhabdom_ri = rri # rhabdom refractive index
        self.snells_law = math.asin(self.cytoplasm_ri / self.rhabdom_ri) / self.conv # calculate angle for total internal reflection using Snell’s law
        self.critical_angle = 90 - self.snells_law # critical angle below which light is totally internally reflected within rhabdom
        self.mx = math.sqrt((self.rhabdom_length ** 2) + (self.rhabdom_radius ** 2)) # mx???

        # output initial P (pigment) and T (tapetum) to output file one
        self.write_output(self.outputfile_one, self.shielding_pigment_length) # write pigment length to output file
        self.write_output(self.outputfile_one, self.reflective_tapetum_length) # write tapetum length to output file

        self.cz = 0     # increases angle of acceptance of rhabdom - initialise to false

        return


    def run_model(self, graphicsflag):
        """Main workhorse of the program. Runs the ray tracing model with the given parameters."""

        # print start_time and write to debug file
        start_time = time.time()
        self.write_output(self.debug_file, "********************\n%s\n" % date.fromtimestamp(start_time).strftime("%d/%m/%Y %H:%M:%S"))

        # do the initial calculations
        self.initial_calculations()

        # main program loop
        while True:
            # calculate prox-dist length of first pass
            if self.boa > self.critical_angle and self.boa < 25 and self.cz == 0:
                # change shape of proximal portion of the rhabdom
                self.boa -= self.proximal_rhabdom_angle
            if self.inter_ommatidial_angle == 0:
                # ray absorbed by proximal shielding pigment
                self.case_four()
            else:
                self.y = self.rhabdom_radius / math.tan(self.boa * self.conv)
                if self.y >= self.rhabdom_length:
                    # ray reflected off base of rhabdom by tapetum
                    self.case_three()
                elif self.y > self.rhabdom_length - self.shielding_pigment_length or self.y > self.rhabdom_length - self.reflective_tapetum_length or self.boa < self.critical_angle:
```

```
                    self.case_two()
                else:
                    # light passes through rhabdom
                    self.case_one()
                    # goto 1002
                    if self.rhabdom_length <= self.refl
e_tapetum_length or self.rhabdom_length <= self.shielding_pigment_length:
                        pass
                    else:
                        # *** call display graphics
 ***
                        continue

            self.current_facet += 1
            self.rhabdom_radius = self.rhabdom_width / 2

            self.rhabdom_length = self.old_rhabdom_length
            self.inter_ommatidial_angle += self.optical_axis
            self.cz = 0 # set CZ as false

            # row complete append 998 and output to file
            self.col_total = len(self.rowdata)
            self.rowdata.append(998)
            self.write_output(self.outputfile_one, self.rowdata

            # append row to output_data for outputfile_two
            self.output_data.append(self.rowdata[0:-1])
            self.row_total = len(self.output_data)

            # clear self.rowdata
            self.rowdata = []

            # account for refraction at cornea
            if self.inter_ommatidial_angle < 60:
                self.boa = (self.inter_ommatidial_angle * 0
) + 3.38
            if self.inter_ommatidial_angle < 50:
                self.boa = (self.inter_ommatidial_angle * 0
) + 0.8676
            if self.inter_ommatidial_angle < 35:
                self.boa = (self.inter_ommatidial_angle * 0
) + 0.1648
            if self.inter_ommatidial_angle < 15:
                self.boa = (self.inter_ommatidial_angle * 0
) + 0.004667
            if self.inter_ommatidial_angle > 60:
                self.print_output("*** UNREAL ANGLE AT CORNE
")
                self.write_output(self.outputfile_one, "UNR
ANGLE AT CORNEA")

            # light loss at cone due to angle of incidence
            self.cc = self.facet_width / math.tan(self.boa * se
nv) # CC???
            if self.cc > (self.facet_width * 2):
                self.fw = math.cos(self.inter_ommatidial_an
 self.conv) * self.facet_width # FW???
            else:
                self.ll = ((2 * self.cc) - (2 * self.facet_
))
                self.fw = math.sin(self.inter_ommatidial_an
 self.conv) * self.ll
            self.facet_num = self.fw / self.facet_width

            # account for change in angle between adjacent rhab
            self.fd = self.num_facets / self.blur_circle_extent
```

```
??? - this is used to divide the aperture up
                        # if current facet is outside edge of eyeshine patch the
n break out of for loop
                        # otherwise check where the current facet is and transfe
r POL to appropriate rhabdom accordingly
                        self.nx = 1
                        for i in range(self.blur_circle_extent):
                                if self.current_facet >= self.num_facets:
                                        pass
                                elif self.current_facet >= (self.fd * self.nx):
                                        self.boa += self.optical_axis
                                        self.rowdata.append(0)
                                self.nx += 1

                        # check to see if edge of eyeshine patch has been reache
d
                        if self.current_facet >= self.num_facets:
                                pass
                        else:
                                continue

                        # iterate over output data
                        for col in range(self.col_total):
                                for row in range(self.row_total):
                                        if self.col_total > len(self.output_data
[row]):
                                                for i in range(self.col_total -
len(self.output_data[row])):
                                                        self.output_data[row].ap
pend(0)
                                        # check all rows
                                        if self.output_data[row][col] > 0:
                                                self.ab[col] = 1 - math.exp(-0.0
067 * self.output_data[row][col])
                                        elif self.output_data[row][col] == 0:
                                                self.ab[col] = 0
                                        if col == 0 and self.ab[col] > 0:
                                                self.bx = 100 * self.ab[col]
                                        if col > 0 and self.ab[col] > 0:
                                                self.bx = 100 * ((1 - self.tot)
* self.ab[col])
                                        if self.ab[col] == 0:
                                                self.bx = 0
                                        self.tot += (self.bx / 100)
                                        self.aa[col] += self.bx
                                        self.bx = 0
                                        self.write_output(self.debug_file, [col
+ 1, self.aa[col], self.ab[col], row + 1])
                                self.bx = 0
                                self.tot = 0

                        self.x = 0
                        output_tmp = []
                        output_tmp.append(self.reflective_tapetum_length)
                        output_tmp.append(self.shielding_pigment_length)
                        for i in range(self.col_total):
                                self.aa[i] = int(self.aa[i] / self.row_total)
                                output_tmp.append(self.aa[i])
                                self.aa[i] = 0
                        output_tmp.append(999)
                        self.print_output("")
                        self.write_output(self.outputfile_two, output_tmp)
                        self.bx = 0

                        # reset tapetum to zero and increase pigment by 10%
                        if self.reflective_tapetum_length >= self.max_rhabdom_le
```

```
ngth and self.shielding_pigment_length >= self.max_rhabdom_length:
                                # increment iteration count and output coun
screen and 999 to the file
                                self.iteration_count += 1
                                self.write_output(self.outputfile_one, 999)
                                # end of program
                                sys.stdout.write("\a") # beep
                                sys.stdout.flush() # flush beep
                                break
                        elif self.reflective_tapetum_length >= self.max_rha
length and self.shielding_pigment_length < self.max_rhabdom_length:
                                self.reflective_tapetum_length = 0
                                self.shielding_pigment_length += self.incre
amount
                        else:
                                self.reflective_tapetum_length += self.incr
_amount

                        # increment iteration count and output count to scr
nd 999 to the file
                        self.iteration_count += 1
                        self.write_output(self.outputfile_one, 999)

                        # reset output data
                        self.output_data = []

                        # reset parameters
                        self.reset_parameters()

                # print end_time
                end_time = time.time()
                self.write_output(self.debug_file, "\n%s\n********************" %
e.fromtimestamp(end_time).strftime("%d/%m/%Y %H:%M:%S"))

        def case_one(self):
                # no reflection - light passes through rhabdom
                self.x = self.rhabdom_radius / math.sin(self.boa * self.con
                self.rowdata.append(self.x * self.facet_num)
                self.rhabdom_length -= self.y
                self.boa += self.optical_axis
                self.cz = 1 # set CZ to true


        def case_two(self):
                # reflection from edge
                self.x = self.rhabdom_radius / math.sin(self.boa * self.con
                self.z = (self.rhabdom_length - self.y) / math.cos(self.boa
lf.conv)

                if self.z > self.x:
                        self.z = self.x
                if (self.x + self.z) > self.old_rhabdom_length:
                        self.v = self.x + self.z
                elif (self.x + self.z) < self.old_rhabdom_length:
                        self.v = self.old_rhabdom_length

                if self.reflective_tapetum_length == 0:
                        val = (self.x + self.z) * self.facet_num
                elif self.reflective_tapetum_length > 0:
                        val = (self.x + self.z + self.v) * self.facet_num
                if self.shielding_pigment_length > 0:
                        val = (self.x + self.z) * self.facet_num
                if self.shielding_pigment_length > (self.rhabdom_length - s
):
                        val = self.x * self.facet_num
                self.rowdata.append(val)
                return

        def case_three(self):
                # bounce off base
                if self.y == self.rhabdom_length:
                        self.x = self.mx
                if self.y > self.rhabdom_length:
                        self.x = self.rhabdom_length / math.cos(self.boa * self.conv)
                if self.x > self.old_rhabdom_length:
                        self.v = self.x
                if self.x < self.old_rhabdom_length:
                        self.v = self.old_rhabdom_length

                if self.reflective_tapetum_length == 0:
                        val = self.x * self.facet_num
                if self.reflective_tapetum_length > 0:
                        val = (self.x + self.v) * self.facet_num
                if self.shielding_pigment_length > 0:
                        val = self.x * self.facet_num
                self.rowdata.append(val)
                return

        def case_four(self):
                # perpendicular ray
                if self.reflective_tapetum_length > 0:
                        val = (self.rhabdom_length * 2) * self.facet_num
                if self.reflective_tapetum_length == 0:
                        val = self.rhabdom_length * self.facet_num
                if self.shielding_pigment_length > 0:
                        val = self.rhabdom_length * self.facet_num
                self.rowdata.append(val)
                return

        def setup_files(self, sn):
                """Setup the filenames and remove old ones if they exist."""
                # get current directory and build filenames
                species_name = sn.lower() # always convert to lowercase
                curr_dir = os.getcwd() # get current working directory
                self.outputfile_one = os.path.join(curr_dir, species_name + ’_output_one.csv’) # outputfile one
                self.outputfile_two = os.path.join(curr_dir, species_name + ’_output_two.csv’) # outputfile two
                self.matrixfile_one = os.path.join(curr_dir, species_name + ’_summary_one.csv’) # matrixfile one
                self.matrixfile_two = os.path.join(curr_dir, species_name + ’_summary_res.csv’) # matrixfile two
                self.matrixfile_three = os.path.join(curr_dir, species_name + ’_summary_sen.csv’) # matrixfile three
                self.debug_file = os.path.join(curr_dir, species_name + ’_debug.txt’) # debug file

                # check if files exist and delete them
                if os.path.exists(self.outputfile_one):
                        os.remove(self.outputfile_one)
                if os.path.exists(self.outputfile_two):
                        os.remove(self.outputfile_two)
                if os.path.exists(self.matrixfile_one):
                        os.remove(self.matrixfile_one)
                if os.path.exists(self.matrixfile_two):
                        os.remove(self.matrixfile_two)
                if os.path.exists(self.matrixfile_three):
                        os.remove(self.matrixfile_three)
                if os.path.exists(self.debug_file):
                        os.remove(self.debug_file)
                return
```

```

        def write_output(self, filename, data):
                """Write data to an output filename."""
                # open file for append and write data
                filehandle = open(filename, ’a’) # open file in append mode
                if isinstance(data, list):
                        csv_data = ",".join(map(str, data))
                else:
                        csv_data = str(data)
                filehandle.write(csv_data + "\n") # write output_text string
file with new line character
                filehandle.close() # close file
                return

        def print_output(self, text):
                """Output text and progress information to the screen."""
                print "%d: (T:%0.2f P:%0.2f) %s" % (self.iteration_count, self.re
tive_tapetum_length, self.shielding_pigment_length, text)
                return

        def reset_parameters(self):
                """Reset all the parameters to their default values."""
                # get stored parameters
                (sn, rl, rw, ed, fw, ad, cri, rri, bce, pra) = self.eye_par
rs

                # reset eye parameters using stored values
                self.num_facets = 0 # num of facets across aperture
                self.rhabdom_width = rw # rhabdom width/diameter
                self.aperture_diameter = ad # aperture diameter
                self.y = 0 # y??? - set to one originally, but we use 0 bas
dexing in python
                self.facet_width = fw # facet width
                self.eye_diameter = ed # eye diameter

                # do the initial calculations
                self.initial_calculations()
                return

        def return_parameters(self):
                """Get the original parameters, as stored at the beginning of the program."""
                # get stored parameters
                (sn, rl, rw, ed, fw, ad, cri, rri, bce, pra) = self.eye_par
rs

                # return parameters to user
                return rl, rw, ed, fw, ad, cri, rri, bce

        def summarise_data(self):
                """Summarise the data produced by the calculations in the run_model function."""
                # get stored parameters
                (sn, rl, rw, ed, fw, ad, cri, rri, bce, pra) = self.eye_par
rs

                # set required parameters
                self.facet_width = fw
                self.eye_diameter = ed
                self.eye_circumference = (22.0 / 7.0) * float(self.eye_diam
 # need 22.0 / 7.0 here as rounds down to 3 with being an integer
                self.inter_ommatidial_angle = (self.facet_width / self.eye_
mference) * float(360)
                self.reflective_tapetum_length = 0
                self.shielding_pigment_length = 0
                self.absorbance = 0
                self.facet = 0
                self.rhabdom = 0
                self.rhabdoms = 21*[0]
                self.tot = 0
                self.bx =       0
                self.torus = 0
                self.inci = 0
                self.area = 0
                self.arem = 0
                self.sens = 0
                self.rhab = 0
                self.rens = 0
                self.cc = 0
                self.dd = 0
                self.frac = 0
                self.oab = 0
                self.matrix_sens = []
                self.matrix_rhab = []
                self.matrix_res = []

                # setup outputfile filehandle
                filehandle = open(self.outputfile_one, ’r’)

                # iterate over file
                for line in filehandle.readlines():
                        line = line.rstrip()
                        if not line:
                                break
                        if re.match("^([0-9]+\.[0-9]{1,})$", line):
                                if self.reflective_tapetum_length == 0:
                                        self.reflective_tapetum_length = float(line)
                                elif self.shielding_pigment_length == 0:
                                        self.shielding_pigment_length = float(line)
                        elif re.match("^([0-9\.\,\s]+998)$", line) and line != "999":
                                text = re.sub("\s+", "", line)
                                parts = text.split(’,’)
                                for part in parts:
                                        # check if end of line
                                        if part == "998":
                                                self.rhabdom = 0
                                                self.bx = 0
                                                self.tot = 0
                                                self.facet += 1
                                                self.area = self.pi * (self.facet + 0.5) ** 2
                                                if self.facet == 0:
                                                        self.torus = self.pi * (0.5) ** 2
                                                self.inci = self.pi * (self.facet - 0.5) ** 2
                                                self.torus = self.area - self.inci
                                                if self.area > self.arem:
                                                        self.arem = self.area
                                        else:
                                                part = float(part) # convert to float for calculations
                                                if part > 0:
                                                        self.absorbance = 1 - math.exp(-0.01 * part) # calculate absorbance
                                                else:
                                                        self.absorbance = 0 # light doesn’t strike rhabdom
                                                if self.rhabdom == 0 and self.absorbance > 0:
                                                        self.bx = (100 * self.ab
```

```
sorbance) # axial rhabdom
                                                elif self.rhabdom > 0 and s
bsorbance > 0:
                                                        self.bx = (100 * ((
elf.tot) * self.absorbance))
                                                if self.absorbance == 0:
                                                        self.bx = 0 # bx =
 not absorbed
                                                self.tot += (self.bx / 100)
                                                self.bx *= self.torus
                                                for i in range(len(self.rha
)):
                                                        if self.rhabdom ==
                                                                self.rhabdo
 += self.bx
                                                self.rhabdom += 1 # increme
abdom
                                                self.bx = 0
                        elif line == "999":
                                # finished block of numbers - work out abso
n
                                self.rhabdom = 0
                                self.sens = sum(self.rhabdoms)
                                self.rhab = self.rhabdoms[0] / self.sens
                                self.halfway_point = self.rhabdoms[0] / 2
                                self.xz = self.rhabdoms[0]
                                self.yy = self.rhabdoms[1]
                                self.optic_axis = 0
                                for i in range(1, 12):
                                        if self.halfway_point < self.rhabdo
:
                                                self.xz = self.rhabdoms[i]
                                                self.yy = self.rhabdoms[i+1
                                                self.optic_axis = self.inte
atidial_angle * i
                                self.diff = self.xz - self.yy
                                self.hwp = self.xz - self.halfway_point
                                self.frac = self.hwp / (self.diff + 0.1)
                                self.oab = self.frac * self.inter_ommatidia
le
                                self.res = self.oab + self.optic_axis # wid
 50% point
                                for i in range(16):
                                        self.rhabdoms[i] = int(self.rhabdom

                                if self.cc == 0:
                                        self.matrix_sens.append(0)
                                        self.matrix_rhab.append(0)
                                        self.matrix_res.append(0)
                                        self.write_output(self.matrixfile_t
 self.matrix_sens)
                                        self.write_output(self.matrixfile_o
elf.matrix_rhab)
                                        self.write_output(self.matrixfile_t
elf.matrix_res)
                                        self.matrix_sens = []
                                        self.matrix_rhab = []
                                        self.matrix_res = []
                                self.matrix_sens.append(int(self.sens / sel
m))
                                self.matrix_rhab.append(int(self.rhab * 100
                                self.matrix_res.append(int(self.res * 200))
                                self.print_output("CC: %s DD: %s" % (str(self
 str(self.dd)))
                                self.iteration_count += 1
                                self.cc += 1
                                if self.cc == 11:
                                        self.dd += 1
                                        self.cc = 0
                                self.rhabdoms[0] = 0
                                self.rhabdoms[-1] = 0
                                self.bx = 0
                                self.facet = 0
                                self.reflective_tapetum_length = 0
                                self.shielding_pigment_length = 0
                self.write_output(self.matrixfile_three, self.matrix_sens)
                self.write_output(self.matrixfile_one, self.matrix_rhab)
                self.write_output(self.matrixfile_two, self.matrix_res)
                self.matrix_sens = []
                self.matrix_rhab = []
                self.matrix_res = []

                # close filehandle
                filehandle.close()

                # let user know we've finished
                # end of program
                sys.stdout.write("\a") # beep
                sys.stdout.flush() # flush beep
                self.print_output("*** End of program ***")

                return

        def build_plots(self):
                """This function will produce publication quality plots from the output data."""
                return

# check for main subroutine and call it
if __name__ == "__main__":
        sys.exit(main())
```