```
"""A module dedicated to evaluating horizontal and vertical positions of the
points of intersection between an array of incident rays and a specified
surface. Also evaluates the clockwise angles (in radians) with repsect to the
vertical of the refracted rays."""

USER = "Jacky Cao"
USER_ID = "bbvw84"

#Numpy is imported to this module as it's various functions are required
throughout
import numpy

#Functions
def refraction_2d (incident_rays, planar_surface):
    '''A function which can calculate the location of a refracted ray and also
the angle the refracted ray makes with the vertical. Input requires is an array
for both incident rays and planar surface.'''

    #Let's define the point of intersection early, xa as x-coord and ya as y-
coord
    a = incident_rays
    at = a[:,2] #angle clockwise from the vertical at the initial point

    #We shall just worry about the planar surface first and the angle that
creates
    b = numpy.array(planar_surface) #1-dimensional list of data
    b.shape = (1,6)

    #Indexing all the various values of the planar surface array
    ps_h1 = b[0,0] #horizonatal coord of one end of the line (the 2d surface)
(m)
    ps_v1 = b[0,1] #the corresponding vertical coord (m)
    ps_h2 = b[0,2] #the horizontal coord of the other line (m)
    ps_v2 = b[0,3] #the corresponding vertical coord (m)
    ps_r1 = b[0,4] #the refractive index of the medium on the incident side of
the surface (no units)
    ps_r2 = b[0,5] #the refractive index of the medium on the refracted side of
of the surface (no units)

    #Working out the angle the plane is tilted at:
    v1 = ps_h1 - ps_h2
    h1 = ps_v1 - ps_v2
    ap = numpy.arctan((v1/h1)) #the angle the plane is at
    #ap2 = numpy.arctan((- v1/h1))
    #print ap
    #print ap2

    #gradient of the plane
    mp = (ps_v2 - ps_v1)/(ps_h2 - ps_h1)
    """print "mp:"
    print mp"""

    #y-intercept of the planar equation
    cp = (- ps_h1 * mp) + ps_v1
    """print "cp:"
    print cp"""

    #the gradient of the incident ray
```

```
    mr = numpy.tan((numpy.pi / 2) - incident_rays[:,2])
    """print "mr:"
    print mr"""

    #y-intercept of the incident ray
    cr = (- a[:,0] * mr) + a[:,1]
    """print "cr:"
    print cr"""

    #calculating x-coord of point of intersection with plane
    xa = (cp - cr)/(mr - mp)
    """print "xa:"
    print xa"""

    #calculating y-coord of the point of intersection with plane
    ya = ((cr * mp) - (cp * mr)) / (mp - mr)
    """print "ya:"
    print ya"""

    #So we are trying to intially calculate the angle of incidence

    """ #do not need I don't think but I'll still keep it
    ratio_r2r1 = ps_r2 / ps_r1

    if ratio_r2r1 >= 1:
        print "Ratio of the two refractive indexes is greater than one, the
ratio must be equal or less than one for the calculation of the critical angle
to be valid."
        print "Thus the function will now be terminated."
        return
    """

    numpy.seterr(all = 'ignore') #getting a runtime warning for an invalid error
encountered in arcsin, can't be bothered dealing with it so I got rid of it
    critical_angle = numpy.arcsin((ps_r2 / ps_r1)) #this gives an impossible
value but I'm not really that knowledged to think around it so I just accept it
    #print critical_angle #test

    #function terminator if the angles given are greater than the critical angle
    if any(a[:,2]) >= critical_angle:
        print "Hawk! An exception has been rasied:"
        print "An incident angle (or multiple angles) is greater than critical
angle, function has been terminated. (maybe, not sure, function still has been
terminated)"
        return
    else:
        print "Incident angles are valid, function will proceed. (maybe)"
#'useful' text-based feedback to show if the function is actually working or not

    #Calculating the angle of incidence
    """ap2 = (numpy.pi / 2) - ap
    at2 = (numpy.pi) - at
    ai2 = (numpy.pi) - at2 - ap2
    print "ai2:"
    print ai2""" #old code, didn't know what I was doing

    ai = (numpy.pi / 2) - a[:,2] - ap
    """print "ai:"
```

```
    print ai"""

    #Snell's law for the angle of refraction
    #ar = numpy.arcsin((ps_r1 / ps_r2) * numpy.sin(ai)) + mp
    """print "ar:"
    print ar"""

    ar1 = numpy.arcsin((ps_r1 / ps_r2) * numpy.sin(ai))
    #ar = (numpy.pi) + ar1 - ai

    ar = ap + (numpy.pi / 2) - ar1

    #print ar

    #Outputting final array for refracted rays
    rr = numpy.array([xa, ya, ar]) #the rays are shown vertically
    rr2 = numpy.rot90(rr, 3) #next two lines of code are to make it look the
same as the initial incident rays array
    refracted_rays = numpy.fliplr(rr2)

    """
    print "refracted_rays:"
    print refracted_rays
    """

    return refracted_rays

#Testing code here
if __name__ =='__main__':

    print "beginning of test" #Useful to show me where code is actually being
tested and if anything from the function has 'leaked' out

    import ray_mini #should be a self contained module, apart from the inital
data - as shown below
    import numpy

    #test data
    #incident_rays =
numpy.array([[0.0,5.0,numpy.pi/2.0],[0.0,5.0,1.1*numpy.pi/2.0]])
    #planar_surface = numpy.array([5.0,2.0,6.0,8.0,1.0,1.33])


ray_mini.refraction_2d(numpy.array([[0.0,5.0,numpy.pi/2.0],[0.0,5.0,1.1*numpy.pi
/2.0]]), numpy.array([4.5, 2.0, 5.5, 8.0,1.0,1.33]))

    #incident_rays = numpy.array([[1.0,0.0,numpy.pi],[1.0,0.0,numpy.pi]]) #more
test data
    #planar_surface = numpy.array([0.0,0.0,3.0,0.0,1.33,1.0])

    #print "Printing incident_rays"
    #print incident_rays

    #ray_mini.refraction_2d(incident_rays, planar_surface) #calling the function
from the imported module

    print "end of test"
```