Introduction to Programming in Python

Mini-Project

(Module: PHYS1101: Discovery Skills in Physics – 2015-2016)

Submission Instructions

To submit your program, start DUO, go to the Discovery Skills in Physics module, click on *Assignments* in the left-hand side column and select *Programming MiniProject*. Attach one (and only one) file. This should be your mini-project module file containing the function(s) that you have written. **Call the file ray_mini.py**. Please do not write any comments in the comment box. When you are ready to submit your work, click on the *Submit* button at the bottom of the page. If you later improve your module you can submit an updated version. (**Beware**: The last version you submit will be marked).

At the top of your file (after your module documentation string), you must put two variables with your name and CIS login as follows:

USER = "Put your name here as a string"

USER_ID = "Put your CIS login here as a string"

The two variable names must be exactly as shown (upper case). These variables will be read and used by the marking program. The mini-project is due by **18.00 on Friday 18 December.**

Assessment

The mini-project will be assessed **formatively**.

Your module will be marked during your normal workshop session in the first week of the Epiphany term. Your work will be marked one-to-one by a demonstrator. You **must** attend for this marking and **bring a hard copy of your code** for written feedback. The demonstrator will test the module that you submitted by importing it into a standard test program and running that program. The test program will call your function with some standard test data. If your function does not meet the specifications given below, the test program may crash and you will lose marks.

It is important to **name functions exactly as instructed in this script** or they will not be found by the automatic marking process and you may lose marks.

Arguments (parameters) to functions should be accepted **in the same order as defined in this script** or they will be incompatible with the automatic marking process and you will lose marks.

Marking Scheme

For convenience, the mini-project will be marked out of 10 with marks awarded as follows:

Module imports correctly: 1 mark

Module produces correct output for test cases: 4 marks

Module and function documentation strings are present and relevant: 2 marks

Good programming techniques and style including comments: 3 marks

Level 1 python mini-project: Ray Tracing

Task

It has been proposed that the task of optical design using *ray tracing* could be made more efficient and flexible using python and numpy. Our initial task is to write a python/numpy function to determine the paths of a set of monochromatic light rays that have been refracted at a planar surface. This surface is the interface between two transparent media of differing refractive index. We shall initially consider the problem in two dimensions only, so that the planar surface, the incident rays and refracted rays will all appear as a set of straight *lines*. The lines representing *rays* will be specified by a horizontal and vertical position origin of the ray in metres and a clockwise angle in radians from the positive vertical direction. The line representing the planar surface will be represented by the horizontal and vertical positions in metres of the two ends of the line. The initial purpose is to test the speed of the function when written in python with appropriate use of numpy.

Write a python function that evaluates the horizontal and vertical positions of the points of intersection between an array of incident rays and a specified surface. The function must also evaluate the clockwise angles in radians with respect to the vertical of the refracted rays.

The function should return a 2-dimensional `numpy` array of floats that should contain, for each specified light ray (in order), the horizontal coordinate of the point of intersection with the surface (in metres), the corresponding vertical coordinate, and the clockwise angle (in radians) with respect to the positive vertical direction of the direction of the refracted ray. The returned array must be ordered with these three values specified by the second index, and with the number of the ray specified by the first index (following the order of the incident rays). If *any* of the incident rays exceeds the critical angle then your function should raise an exception.

The function name, inputs and outputs should be defined as follows:

```
def refraction_2d (incident_rays, planar_surface):
    '''Docstring'''
# Your code here, which inserts real values
# into the output array refracted_rays.
    return refracted_rays
```

The arguments and outputs are defined as follows:

- `incident_rays` – a 2-dimensional numpy array of floats, with the individual incident light ray specified by the first index, and the following values specified by the second index for each ray (in order): the horizontal coordinate (in metres) of the point of origin of the ray, the corresponding vertical coordinate, and the clockwise angle (in radians) with respect to the positive vertical direction of the direction of the incident ray.

  Note: An illustrative example value of incident_rays for testing your function might be:-
  `incident_rays=numpy.array([[0.0,5.0,numpy.pi/2.0],[0.0,5.0,1.1*numpy.pi/2.0]])`
  which has shape (2,3), and specifies 2 rays (in this example):
  `[0.0, 5.0, numpy.pi/2.0]` specifies ray 0, which we could access as `incident_rays[0]`
  `[0.0, 5.0, 1.1*numpy.pi/2.0]` specifies ray 1, which we could access as `incident_rays[1]`
  So in order to access the angle of the second ray, for example, we would use `incident_rays[1,2]`

- `planar_surface` – a 1-dimensional numpy array of floats, specifying (in order): the horizontal coordinate (in metres) of one end of the line (the 2-dimesional "surface"), the corresponding vertical coordinate, the horizontal coordinate of the other of the line, the corresponding vertical coordinate, the refractive index of the medium on the incident side of the surface, and the refractive index of the medium on the other side of the surface.

- `refracted_rays` – a 2-dimensional numpy array of floats, with the individual refracted light ray specified by the first index (in the same order as the incident rays) and the following values specified by the second index for each ray (in order): the horizontal coordinate (in metres) of the point of intersection of the corresponding incident ray with the specified planar surface, the corresponding vertical coordinate, and the clockwise angle (in radians) with respect to the positive vertical direction of the direction of the refracted ray.

Simplifications

You do NOT have to worry about:

1. Rays that miss the surface. All of the test rays specified in the marking program will hit the specified surface.

2. Rays that have clockwise angles to the positive vertical axis of greater than or equal to pi radians. All of the test rays will all have angles less than this.

3. Surfaces or rays with infinite gradients

Testing

Test your code thoroughly before submitting your work. Do this by calling your function with sample input numbers and confirming that the output is what you expect.

Note that your code should behave as a module. If it is imported, no code in the module should be executed unless a function in the module is explicitly called. If you want to include code that runs when the python file is executed you should do this in a "test block" as follows:

```
if __name__=='__main__':
     # Your test code indented here
```

You may submit your module with test code included, as long as it is in a test block. Your test code will not be marked.

Please note that your project will be marked by an automatic marking program that will import your module and call your function with some test values. It will test the execution speed and the accuracy of the returned results.