420-SN1-RE Programming in Science - Lab Exercise 6

September 13, 2024

Introduction

Goals for this lab:

- Practice using imported functions.

In this lab you will create programs to compute certain results that require the use of functions imported from common Python modules.

Exercise 1 - Great circle distance

The shortest distance between two points on a spherical surface (such as that of Earth, more or less) is measured along the smaller of two Great Circle arcs that connects the two points.

Any two chosen points on Earth's surface and the center of Earth determine exactly one plane. This plane intersects the Earth's surface along a circle, which is called the *great circle*, because the intersecting plane goes through the center and therefore gives the largest circle of intersection with the sphere. For any two points on Earth's surface, there will be two great circle arcs connecting the points. The shorter of the two (they may be equal) gives the shortest distance between the points as measured on Earth's surface. This particular calculation ignores both the altitude of the airplane and the fact that the Earth's surface is not a perfect sphere.

- **Latitudes:** Horizontal lines that measure distance north or south of the equator. Negative latitudes for locations south of the Equator.
- **Longitudes:** Vertical lines that measure east or west of the meridian in Greenwich, England. Negative longitudes for locations west of the 0 degree meridian.
- **Ex-1:** Montreal has a negative longitude (between 73 and 74 degrees west) and positive latitude (a little over 45 degrees north).
- **Ex-2:** Rio de Janeiro, Brazil, both coordinates are negative.

What you have to do:

- Use the program skeleton provided, `lab06-p1.py` to complete a Python program that computes the great circle distance between two given locations.
- The skeleton has the basic input statements you will need to use. Add your code to this file.

- You have to convert the coordinates to radians using the `radians()` function imported from `math`.
- You will also import `sin()`, `cos()`, and `acos()`.
- Remember that $\cos^{-1}(x)$ is written `acos(x)` in Python.

The formula: The computed distance (d) is given by the formula:

$$d = \alpha R$$

where $R = 6371$ km, the mean radius of the Earth. The value α is given by the formula:

$$\alpha = \cos^{-1}\left(\sin(t_1)\sin(t_2) + \cos(t_1)\cos(t_2)\cos(n_1 - n_2)\right)$$

where t_1 and n_1 are, respectively, the latitude and longitude of the first point, and t_2 and n_2 are the latitude and longitude of the second point.

Test cases

Test your program by entering two pairs of real coordinates of your choice. That is, print out the distances between two different pairs of locations.

Test Case 1: Marianopolis College is at 45.4814, -73.6115, and the Vancouver Aquarium is at 49.3006, -123.1309. The great circle distance between these points is 3684 km.

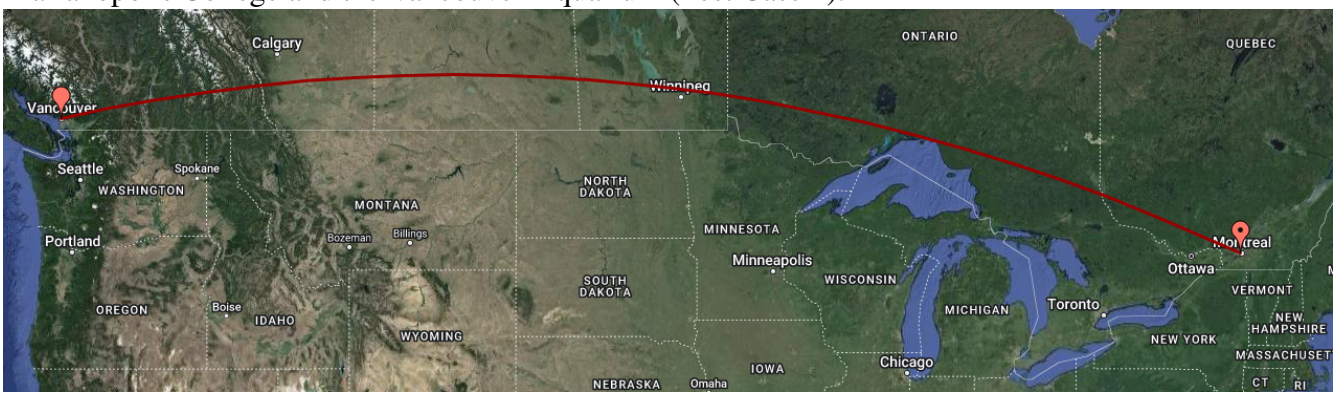
Test Case 2: St. John's, NFL is at 47.5702, -52.6819 and Tromsø, Norway is at 69.6480, 18.9606 for a distance of 4449 km.

You can get sample latitude and longitude numbers in the appropriate format using Google Maps: https://support.google.com/maps/answer/18539?hl=en

You can check your work against this website, which uses a slightly more complex calculation to maintain accuracy over short distances:

http://www.movable-type.co.uk/scripts/latlong.html

Here is an illustration of the great circle distance generated by this website, using the end points of Marianopolis College and the Vancouver Aquarium (Test Case 1).



Exercise 2 - Drawing a triangle

Python includes several tools for drawing pictures and charts. One of the simplest is called `turtle`. Turtle graphics was created as part of the Logo programming language in 1967. While Logo is no longer widely used, the turtle graphics concept has been implemented in many other languages.

The idea is that you are issuing commands to a robot or "turtle" that can draw by dragging a pen across a sheet of paper. **The robot starts in the center of the page, facing right.** You can issue commands to tell the robot to go forward or backward, turn left or right, change pen color, and raise or lower the pen. The "turtle" we use is drawing on a screen rather than a sheet of actual paper, but the idea is the same.

While we will use `turtle` in some of these examples, note that you do *not* need to memorize any `turtle` functions.

You will write a program to draw a triangle (Lab06-p2.py). Your program must ask the user to enter three side lengths, using pixel units. Your program will then compute the appropriate angles. Finally, your program will draw the triangle with the appropriate sides and angles. The *orientation* of your triangle will be arbitrary, but the shape should respect the user's inputs.

Geometry

Given a triangle with sides a, b, c and angles α, β, γ. We define α as the angle opposite side a, β as the angle opposite side b, and γ as the angle opposite side c. Given these assumptions, the angles can be derived from the side lengths by the following equations:

$$\alpha = \cos^{-1}\left(\frac{b^2 + c^2 - a^2}{2bc}\right) \tag{1}$$

$$\beta = \cos^{-1}\left(\frac{a^2 + c^2 - b^2}{2ac}\right) \tag{2}$$

$$\gamma = \cos^{-1}\left(\frac{a^2 + b^2 - c^2}{2ab}\right) \tag{3}$$

Note that the inverse trigonometric functions such as $\cos^{-1}(x)$ and $\sin^{-1}(x)$ are written `acos(x)` and `asin(x)` in Python. Note also that Python's `math` module uses angles given in radians, while the `turtle` module uses angles given in degrees. The `math` modules defines functions `degrees(x)` and `radians(x)` that convert from radians to degrees or from degrees to radians, respectively.

Using `turtle`

To use `turtle` you need to import it. You can use the functions `left()` or `right()`, to turn a specified number of degrees to the left or right, respectively. You will also use the `forward()` function, which moves the turtle forward a fixed number of pixels. For example, the three-line program:

```
from turtle import *
left(90)
forward(300)
```

should draw a 300-pixel vertical line starting at the center of the window. Here's a list of other functions you may find useful. **You do *not* need to memorize these functions!**

- `forward(x)` - move the turtle forward `x` pixels.

- `backward(x)` - move the turtle backward `x` pixels (`backward(x)` is the same as `forward(-x)`).
- `right(d)` - turn the turtle right `d` degrees.
- `left(d)` - turn the turtle left `d` degrees (`left(d)` is the same as `right(-d)`).
- `goto(x, y)` - move the turtle to a specific point. The center of the drawing window is at 0, 0.
- `pencolor(c)` - set the pen color. Most common colors can be referred to by `str` names, like `'blue'` or `'red'`.
- `fillcolor(c)` - fill closed shapes with the given color, e.g. `fillcolor('yellow')`.
- `penup()` - raise the pen, temporarily stops drawing.
- `pendown()` - lower the pen, resumes drawing.
- `width(n)` - set the pen width to `n` pixels.
- `begin_fill()` - start drawing a filled area.
- `end_fill()` - finish drawing a filled area.
- `circle(r, e)` - draw a arc with radius `r` and angular extent `e`. The center of the arc is `r` units to the left of the turtle position (from the turtle's point of view). If `e` is omitted, a full circle is drawn.
- `done()` - indicates that the drawing is complete. No more commands can be given after calling this function.
- `speed(s)` - set the drawing speed, where 1 is slow and 10 is fast. 0 is the fastest possible drawing speed.

What you have to do:

- Create a file named `lab06-p2.py` to write your program.
- Here are some *optional* issues to consider and improve:
  1. How can you tell if it is actually possible to draw the desired triangle? What can you do if the user enters impossible values?
  2. You can give the triangle color by adding a call to the `color()` function. It looks like this:

     ```
     color('black', 'red')
     ```

     The first color name sets the color of the pen which draws lines, the second color name sets the "fill color" used for closed shapes. The fill color will be used to color the area of the triangle, if you use the `begin_fill()` and `end_fill()` functions before and after drawing the triangle.

Submitting your work

Combine your 2 Python files (Lab06-p1.py & Lab06-p2.py) into a single zip file and upload them to Omnivox. Be sure to submit a *single* file containing all of your work.