Project 5.2 Escape!

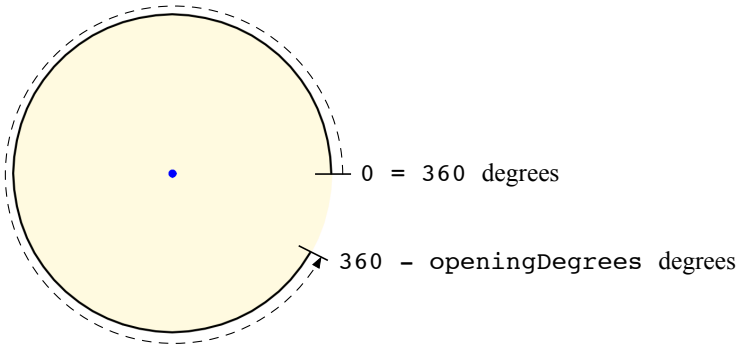
In some scenarios, movement of the "particle" in a random walk is restricted to a bounded region. But what if there is a small opening through which the particle might escape or disappear? How many steps on average does it take the particle to randomly come across this opening and escape? This model, which has become known as the *narrow escape problem*, could represent a forager running across a predator on the edge its territory, an animal finding an unsecured gate in a quarantined area, a molecule finding its way through a pore in the cell membrane, or air molecules in a hot room escaping through an open door.

1. Simulate the narrow escape

Write a function

```
escape(openingDegrees, tortoise, draw)
```

that simulates the narrow escape problem in a circle with radius 1 and an opening of `openingDegrees` degrees. In the circle, the opening will be between 360 – `openingDegrees` and 360 degrees, as illustrated below.



The particle should follow a *normally distributed* random walk, as described in Exercise 5.3.1. The standard deviation of the normal distribution needs to be quite small for the particle to be able to find small openings. A value of $\pi/128$ is suggested in [9]. Since we are interested in the number of steps taken by the particle (instead of the distance traveled, as before), the number of steps will need to be incremented in each iteration of the loop. When the particle hits a wall, it should "bounce" back to its previous position. Since the particle is moving within a circle, we can tell if it hits a wall by comparing its distance from the origin to the radius of the circle. If the particle moves out to the distance of the wall, but is within the angle of the opening, the loop should end, signaling the particle's escape.

Finding the current angle of the particle with respect to the origin requires some trigonometry. Since we know the x and y coordinates of the particle, the angle can be found by computing the arctangent of y/x: $\tan^{-1}(y/x)$. However, this will cause a problem with $x = 0$, so we need to check for that possibility and fudge the value of x a bit. Also, the Python arctangent ($\tan^{-1}$) function, `math.atan`, always returns an angle between $-\pi/2$ and $\pi/2$ radians (between –90 and 90 degrees), so the

result needs to be adjusted to be between 0 and 360 degrees. The following function handles this for you.

```
def angle(x, y):
    if x == 0:        # avoid dividing by zero
        x = 0.001
    angle = math.degrees(math.atan(y / x))
    if angle < 0:
        if y < 0:
            angle = angle + 360   # quadrant IV
        else:
            angle = angle + 180   # quadrant II
    elif y < 0:
        angle = angle + 180       # quadrant III
    return angle
```

Below you will find a "skeleton" of the `escape` function with the loop and drawing code already written. Drawing the partial circle is handled by the function `setupWalls` below that. Notice that the function uses a `while` loop with a Boolean flag variable named `escaped` controlling the iteration. The value of `escaped` is initially `False`, and your algorithm should set it to `True` when the particle escapes. Most, *but not all*, of the remaining code is needed in the `while` loop.

```
def escape(openingDegrees, tortoise, draw):
    x = y = 0                     # initialize (x, y) = (0, 0)
    radius = 1                    # moving in unit radius circle
    stepLength = math.pi / 128    # std dev of each step

    if draw:
        scale = 300               # scale up drawing
        setupWalls(tortoise, openingDegrees, scale, radius)

    steps = 0                     # count of steps taken
    escaped = False               # has particle escaped yet?
    while not escaped:

        # one step of a random walk here
        # if the particle reaches the wall:
        #     if it is in the opening, then exit;
        #     otherwise, "bounce" back to previous saved position

        if draw:
            tortoise.goto(x * scale, y * scale)  # move particle

    if draw:
        screen = tortoise.getscreen()   # update screen to compensate
        screen.update()                 #  for high tracer value
    return steps
```

```
def setupWalls(tortoise, openingDegrees, scale, radius):
    screen = tortoise.getscreen()
    screen.mode('logo')                  # east is 0 degrees
    screen.tracer(5)                     # speed up drawing

    tortoise.up()                        # draw boundary with
    tortoise.width(0.015 * scale)        #   shaded background
    tortoise.goto(radius * scale, 0)
    tortoise.down()
    tortoise.pencolor('lightyellow')
    tortoise.fillcolor('lightyellow')
    tortoise.begin_fill()
    tortoise.circle(radius * scale)
    tortoise.end_fill()
    tortoise.pencolor('black')
    tortoise.circle(radius * scale, 360 - openingDegrees)
    tortoise.up()
    tortoise.home()

    tortoise.pencolor('blue')     # particle is a blue circle
    tortoise.fillcolor('blue')
    tortoise.shape('circle')
    tortoise.shapesize(0.75, 0.75)

    tortoise.width(1)             # set up for walk
    tortoise.pencolor('green')
    tortoise.speed(0)
    tortoise.down()               # comment this out to hide trail
```

2. Write a Monte Carlo simulation

Write a function

```
escapeMonteCarlo(openingDegrees, trials)
```

that returns the average number of steps required, over the given number of trials, to escape with an opening of `openingDegrees` degrees. This is very similar to the `rwMonteCarlo` function from Section 5.1.

3. Empirically derive the function

Write a function

```
plotEscapeSteps(minOpening, maxOpening, openingStep, trials)
```

that plots the average number of steps required, over the given number of trials, to escape openings with widths ranging from `minOpening` to `maxOpening` degrees, in increments of `openingStep`. (The x-axis values in your plot are the opening widths and y-axis values are the average number of steps required to escape.) This is very similar to the `plotDistances` function from Section 5.1.

Plot the average numbers of steps for openings ranging from 10 to 180 degrees, in

10-degree steps, using at least 1,000 trials to get a smooth curve. As this number of trials will take a few minutes to complete, start with fewer trials to make sure your simulation is working properly.

In his undergraduate thesis at the University of Pittsburgh, Carey Caginalp [9] mathematically derived a function that describes these results. In particular, he proved that the expected time required by a particle to escape an opening width of α degrees is

$$T(\alpha) = \frac{1}{2} - 2\ln\left(\sin\frac{\alpha}{4}\right).$$

Plot this function in the same graph as your empirical results. You will notice that the $T(\alpha)$ curve is considerably below the results from your simulation, which has to do with the step size that we used (i.e., the value of `stepLength` in the `escape` function). To adjust for this step size, multiply each value returned by the `escapeMonteCarlo` function by the square of the step size ($(\pi/128)^2$) before you plot it. Once you do this, the results from your Monte Carlo simulation will be in the same *time* units used by Caginalp, and should line up closely with the mathematically derived result.