The following algorithm, described here in plain English, is one possible way to approximate the value of pi using the concept of Buffon's needle. By dropping pins onto a sheet of paper split into quadrants of equal proportion, it is possible to calculate pi by counting the number of pins that cross a vertical line and inserting that value into **Equation 1**.

$$[\boldsymbol{Eqn\ 1}] \qquad \pi = \frac{2 * Length\ of\ Toothpick * Number\ of\ Toothpicks\ Tossed}{Distance\ Between\ Parallel\ Lines * Toothpicks\ Crossing\ a\ Line}$$

For the purposes of this algorithm, three empty spaces demarcated by two equally spaced vertical lines are used. The sheet is dimensioned to 8.4 inches square with each vertical line considered parallel to one another at a perpendicular distance of 2.8 inches. In this model, there are only two lines. The variable **line1** is assigned a value equal to 1/3 the length of one side of the sheet; this may be thought of as the distance from the left edge of the sheet to the "first line." Similarly, the second line, **line2**, is set equal to 2/3 the length of one side of the sheet. The length of the pin is set to 2.5 inches. The variables **z, n, n_ok, and m** are defined as the number of pins that cross a vertical line when fell, the total number of pins dropped *so far*, the number of pins *successfully* dropped onto the sheet, and the *target* total number of drops, respectively.

In order to calculate the value of pi using the Buffon pi method, it is important to simulate a random pin drop. To do this, we randomly define the position of the left-most point on a pin and then the angle formed between the length of the pin and the x-axis. We consider the left and bottom edges of the page to represent the y-axis and x-axis, respectively. The distance from the y-axis is given by variable **x**, which is equal to the product between the length of the sheet and the value produced by a built-in random number generator. The value of **y** is defined *exactly the same*. The reason why **x** and **y** are not equal to one another is because another built-in function seeds the random number generator using the current time. The random number acts as a scalar quantity, which, if greater than one, lands the pin outside the scope of the sheet. The angle between the pin and the x-axis, variable **ang**, is equal to pi times some random number.

With the left-most point on the pin defined, it is possible to identify where the right-most point must lie. More important, however, are the lengths of the horizontal and vertical projections of the pin – whose lines intersect this point. The length of these projections may be directly summed with the distances from the x-axis and y-axis to the left-most point. The total horizontal and vertical distances to the right-most point then are given by **Equations 2** and **3**, respectively.

$$[\boldsymbol{Eqn\ 2}] \qquad x + L * cos(ang) \qquad\qquad [\boldsymbol{Eqn\ 3}] \qquad y + L * sin(ang)$$

With all variables defined and/or initialized, the primary loop that counts whether a pin crosses a line begins. This is a simple *while* loop which will end once the value of **n** is equal to **m**. Every pass of the loop increments **n** by one. There are four *if* statements that will increment **z** by one if their criteria is met. As an example, here is how we know whether a pin has crossed the first line: (1) Is the distance to the left-most point on the pin from the y-axis less than the distance to the first line? *And* (2) Is the distance to the right-most point from the y-axis greater than the distance to the first line? If **yes**, increment **z**. The converse of this criteria *also* indicates that a pin has crossed a line and we represent that with an *Else if* statement; if neither condition is met, then the loop uses the same logic to determine whether the pin has landed across the second line. Whether any of these conditions are met or not, the variable **n** increases by one. The variable **n_ok**, the number of pins dropped *within the sheet*, will only increase if the distance to the right-most point on the pin is greater than 0 and less than the length of the page for both its x and y components. When the loops ends, **Equation 1** is used to approximate pi. The true error and relative true error are then calculated to compare the true value of pi to the value we have approximated.