# Chapel Tutorial Exercise: A Monte Carlo Simulation to Approximate PI

The Monte Carlo simulation to approximate $\pi$ relies on the ratio of the area of a circle to the area of a square where the diameter of the circle is equal to the length of each side of the square:

$$\frac{\pi \cdot r^2}{(2 \cdot r)^2} = \frac{\pi}{4}$$

By computing random points in a square and determining how many of these points are in the circle, a Monte Carlo simulation can be used to approximate the value of $\pi$.

The following serial Chapel code computes an approximation to $\pi$ using a Monte Carlo simulation:

```
use Random;

config const n = 100000;       // number of random points to generate
config const seed = 314159265; // random number generator seed

writeln("Number of points    = ", n);
writeln("Random number seed  = ", seed);

var rs = new RandomStream(seed, parSafe=false);
var count = 0;
for i in 1..n do
  count += rs.getNext()**2 + rs.getNext()**2 <= 1.0;
delete rs;

writeln("Approximation of PI = ", format("#.#######", count * 4.0 / n));
```

Starting with the provided Chapel programs to approximate PI using the Monte Carlo simulation (available in the numbered directories), try your hand at writing the following Chapel codes:

1. **(optional) A Serial Variant.** Using the concepts presented in *Language Basics*, change the provided serial Chapel program to determine the number of points needed to compute an approximation of $\pi$ within a specified value, `epsilon`, of the true value of $\pi$ accurate to 20 decimal places.

2. **A Task-Parallel Version.** Using the concepts presented in *Task Parallelism*, parallelize the provided serial Chapel program. Measure the speedup.

3. **A Multi-Locale Task-Parallel Version.** Using the concepts presented in *Locality and Affinity*, extend the provided task-parallel Chapel program (or your task-parallel program from 2) to run on multiple locales. Measure the speedup across locales, varying the number of tasks per locale.

4. **A Data-Parallel Version.** Using the concepts presented in *Data Parallelism*, parallelize the provided serial Chapel program. Measure the speedup. Compare this code to your task-parallel implementation in terms of performance, effort to write, readability, and maintainability.

5. **A Multi-Locale Data-Parallel Version.** Using the concepts presented in *Distributions and Layouts*, extend the provided data-parallel Chapel program (or your data-parallel program from 4) to run on multiple locales. Measure the speedup across locales. Compare the differences between this code and your single-locale data-parallel code with the differences between your multi-locale and single-locale task-parallel codes.