You can estimate the value of Pi (π) using the Monte Carlo method with a simple geometric approach.[1] Here's how it works:

Conceptual Idea:

1. Imagine a square with sides of length 2r.
2. Inside this square, imagine a circle perfectly inscribed, meaning its diameter is equal to the side length of the square (2r), and its radius is r.
3. The area of the square is (2r) * (2r) = $4r^2$.
4. The area of the circle is π * r^2.
5. The ratio of the circle's area to the square's area is (π * r^2) / ($4r^2$) = π / 4.

Monte Carlo Simulation:

Now, let's use random sampling to approximate this ratio:

1. **Generate Random Points:** Imagine randomly throwing darts at the square. Each dart's landing position can be represented by a pair of coordinates (x, y).
2. **Define the Square:** Let the center of the square (and the circle) be at the origin (0, 0). If the side length of the square is 2r, then the x and y coordinates of any point within the square will fall within the range -r to +r.
3. **Check if the Point is Inside the Circle:** A point (x, y) is inside the circle if its distance from the center (0, 0) is less than or equal to the radius r. We can use the distance formula: sqrt(x^2 + y^2) <= r. Squaring both sides gives us a more efficient check: x^2 + y^2 <= r^2.
4. **Simulate Many Throws:** Generate a large number of random points (x, y) within the square's boundaries.
5. **Calculate the Ratio:** Count how many of these random points fall inside the circle. The ratio of the number of points inside the circle to the total number of points generated will approximate the ratio of the circle's area to the square's area (π / 4).
6. **Estimate Pi:** Multiply this ratio by 4 to get an estimate of Pi: π ≈ 4 * (number of points inside the circle / total number of points).

Python Implementation:

Here's a Python code snippet to demonstrate this:

Python

```
import random
import math
```

```
def estimate_pi(num_points):
    points_inside_circle = 0
    total_points = 0

    for _ in range(num_points):
        # Generate random x and y coordinates between -1 and 1 (assuming r=1)
        x = random.uniform(-1, 1)
        y = random.uniform(-1, 1)

        # Check if the point is inside the unit circle (x^2 + y^2 <= 1)
        if x**2 + y**2 <= 1:
            points_inside_circle += 1
        total_points += 1

    # Estimate Pi
    pi_estimate = 4 * (points_inside_circle / total_points)
    return pi_estimate

# Number of random points to generate
num_samples = 100000

# Estimate Pi
estimated_pi = estimate_pi(num_samples)
print(f"Estimated value of Pi with {num_samples} points: {estimated_pi}")
print(f"Actual value of Pi: {math.pi}")
```

Explanation of the Code:

1. **estimate_pi(num_points) function:**
    - Initializes points_inside_circle and total_points to 0.
    - Iterates num_points times (simulating the dart throws).
    - In each iteration:
        - Generates random x and y coordinates between -1 and 1. This effectively defines a square with side length 2 centered at the origin (we're assuming a radius r=1 for simplicity).
        - Checks if the point (x, y) lies within the unit circle (radius 1) using the condition x**2 + y**2 <= 1.
        - Increments points_inside_circle if the point is inside the circle.
        - Increments total_points for each generated point.
    - Calculates the estimated value of Pi using the formula: 4 * (points_inside_circle / total_points).
    - Returns the estimated Pi.

2. **Main part of the code:**
    - Sets the num_samples to a large number (e.g., 100,000) for better accuracy.
    - Calls the estimate_pi() function to get the estimated value.
    - Prints the estimated value and the actual value of Pi for comparison.

Accuracy:

The accuracy of the Pi estimate using the Monte Carlo method increases with the number of random points generated. As you run the simulation with more and more points, the estimated value will tend to get closer to the true value of Pi. However, it's a probabilistic method, so the result will not be exactly Pi, but rather a statistical approximation.