```
import random
from math import sqrt, pi
from vpython import graph, gdots, color, rate

# INPUT PARAMETERS
x_position = 0.0  # x-coordinate of stone throw
y_position = 0.0  # y-coordinate of stone throw
max_throws = 100  # number of stones to be thrown
circle_radius = 1.0  # radius of the circle
square_side = 2.0  # side length of the square

# Initialization of variables
estimated_pi = 0.0  # variable to hold the estimated value of Pi
inside_count = 0  # counter for stones inside the circle
repeats = 5  # number of repetitions for Pi estimation
total_pi = 0  # accumulator for Pi values over repetitions
average_pi = 0  # the final average of Pi values
inner_iter = 0  # inner loop variable for throws
outer_iter = 0  # outer loop variable for repetitions

# Set up the graphical elements
# Create a graph for plotting Pi values over time
g = graph(title="Monte Carlo Pi Estimation", xtitle="Number of Throws", ytitle="Estimated Pi",
width=600, height=400)

# Plotting dots for stones inside and outside the circle
inside_points = gdots(graph=g, color=color.red)
outside_points = gdots(graph=g, color=color.blue)

# Another graph to plot the Pi estimate
pi_estimate_plot = gdots(graph=g, color=color.black)

# MAIN PROCESSING LOOP
# Repeating the process 'repeats' times for more accurate Pi estimation
for outer_iter in range(1, repeats + 1):
    # Reset variables for each repetition
    inner_iter = 0
    inside_count = 0
    estimated_pi = 0
    # Clear any previous data points from previous runs
    outside_points.delete()
    inside_points.delete()

    # Throwing stones inside the square for a total of 'max_throws' times
```

```
    for inner_iter in range(max_throws):
        rate(10)  # slow down the simulation for better visualization
        # Randomly position a stone inside the square
        x_position = random.random() * square_side - 1  # Random x-coordinate in square
        y_position = random.random() * square_side - 1  # Random y-coordinate in square

        # Check if the stone is inside the circle (using Pythagoras' theorem)
        if sqrt(x_position ** 2 + y_position ** 2) <= circle_radius:
            inside_count += 1  # increment the counter for inside stones
            inside_points.plot(pos=(x_position, y_position))  # plot stone inside the circle in red
        else:  # if the stone is outside the circle
            outside_points.plot(pos=(x_position, y_position))  # plot stone outside in blue

        # Calculate the current estimate of Pi based on the current count
        estimated_pi = 4 * inside_count / (inner_iter + 1)  # Pi ≈ 4 * (inside_count / total_throws)

        # Plot the estimated value of Pi
        pi_estimate_plot.plot(pos=(inner_iter, estimated_pi))

    total_pi += estimated_pi  # accumulate the Pi value for this repetition

# Calculate the final average Pi value after all repetitions
average_pi = total_pi / repeats  # final average Pi value

# OUTPUT RESULTS
print("Estimated Pi =", average_pi)
absolute_error = abs(pi - average_pi)  # Calculate the absolute error from the true Pi value
print("Absolute error =", absolute_error)
```