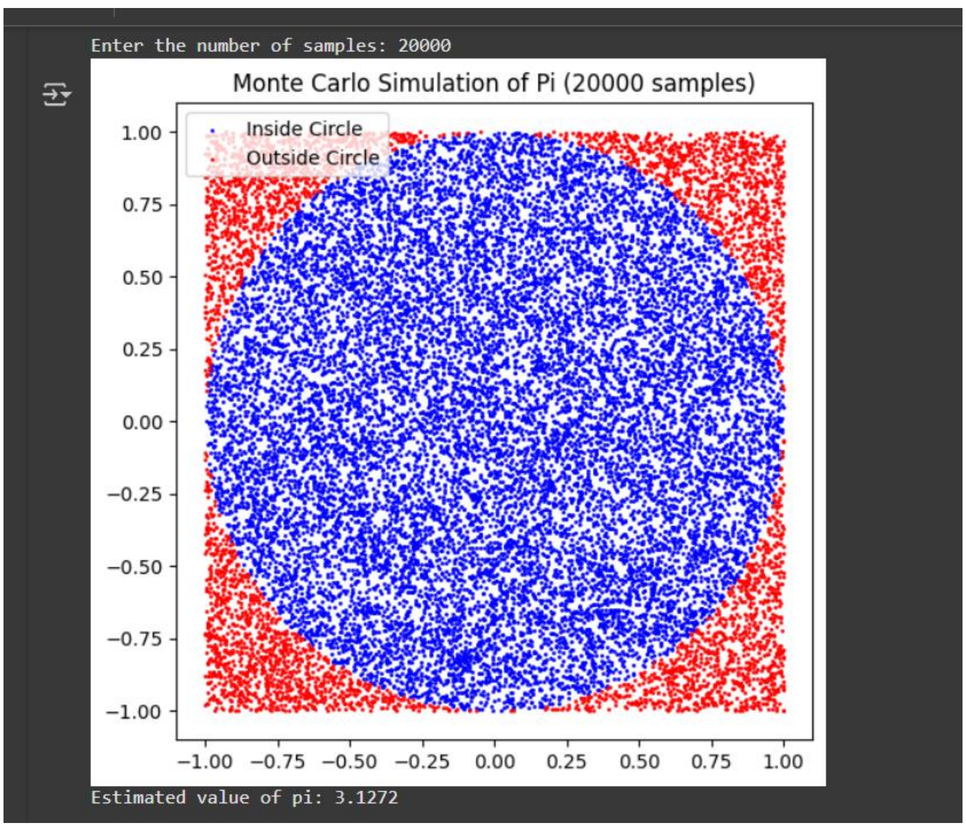
Lab 1: Monte Carlo Simulation method

```
import random

import matplotlib.pyplot as plt


def monte_carlo_pi(num_samples):

    inside_circle = 0

    x_inside, y_inside = [], []

    x_outside, y_outside = [], []


    for _ in range(num_samples):

        x, y = random.uniform(-1, 1), random.uniform(-1, 1)


        if x**2 + y**2 <= 1:

            inside_circle += 1

            x_inside.append(x)

            y_inside.append(y)

        else:

            x_outside.append(x)

            y_outside.append(y)


    pi_estimate = (inside_circle / num_samples) * 4


    plt.figure(figsize=(6,6))

    plt.scatter(x_inside, y_inside, color='blue', s=1, label='Inside Circle')

    plt.scatter(x_outside, y_outside, color='red', s=1, label='Outside Circle')

    plt.gca().set_aspect('equal', adjustable='box')

    plt.legend()
```

```
    plt.title(f"Monte Carlo Simulation of Pi ({num_samples} samples)")
    plt.show()

    return pi_estimate

if __name__ == "__main__":
    num_samples = int(input("Enter the number of samples: "))
    pi_estimate = monte_carlo_pi(num_samples)
    print(f"Estimated value of pi: {pi_estimate}")
```

Output:



Lab 2: Generate a random number using Linear Congruential Method.

```
def linear_congruential_generator(seed=7, num_samples=15, a=1664525, c=1013904223, m=2**32):
    random_numbers = []
    Xn = seed
    for _ in range(num_samples):
        Xn = (a * Xn + c) % m
        random_numbers.append(Xn / m)
    return random_numbers


random_numbers = linear_congruential_generator(seed=50, num_samples=15)


print("Random numbers Generated:")
for num in random_numbers:
    print(f"{num:.5f}")
```

Output:

```
Random numbers Generated:
0.25545
0.81376
0.05495
0.51869
0.70739
0.00735
0.89558
0.44768
0.94353
0.45422
0.48301
0.97817
0.45689
0.47600
0.42308
```

Lab 3: Generate a random number using Mid-Square Method

```
def mid_square_method(seed=4321, num_samples=12):
    random_numbers = []
    Xn = seed
    for _ in range(num_samples):
        Xn = int(str(Xn ** 2).zfill(8)[3:7])  # Square the number, zero-pad, and extract middle digits
        random_numbers.append(Xn / 10000)  # Normalize to [0,1]
    return random_numbers


random_numbers = mid_square_method(seed=6789, num_samples=12)


print("Random numbers Generated:")
for num in random_numbers:
    print(f"{num:.5f}")
```

Output:

```
Random numbers Generated:
0.90520
0.38700
0.76900
0.36100
0.32100
0.04100
0.68100
0.76100
0.12100
0.64100
0.88100
0.16100
```

Lab 4: Test random number using Kolmogorov Smirnov (KS-test)

```
import numpy as np

from scipy import stats

# Generate random numbers from a uniform distribution

sample_size = 50  # Number of random numbers

data = np.random.uniform(0, 1, sample_size)

# Display generated random numbers

print("Generated Random Numbers:")

print(data)

# Perform KS test against a uniform distribution

ks_statistic, p_value = stats.kstest(data, 'uniform')

# Output the results

print(f"\nKS Statistic: {ks_statistic:.5f}")

print(f"P-value: {p_value:.5f}")

# Interpretation of results

alpha = 0.05  # Significance level

if p_value > alpha:

print("Fail to reject the null hypothesis: Data follows the uniform distribution.")

else:

print("Reject the null hypothesis: Data does not follow the uniform distribution.")
```

Output:

```
Generated Random Numbers:
[0.19075336 0.63970091 0.6823643  0.05844876 0.20015052 0.19804052
 0.03754981 0.05119523 0.06130408 0.42631076 0.60058075 0.28654214
 0.81261521 0.84059275 0.82723142 0.97502011 0.04825286 0.66398925
 0.07589803 0.46425239 0.04708441 0.06960733 0.88757109 0.75776637
 0.8435056  0.87453597 0.91134061 0.39198929 0.64068071 0.40235446
 0.21754978 0.85948864 0.6951878  0.1186583  0.33431077 0.34403235
 0.81434854 0.04976299 0.65056356 0.88828254 0.68300601 0.83257276
 0.80456663 0.39170475 0.33393659 0.7109712  0.22583986 0.80071298
 0.70360607 0.68251535]

KS Statistic: 0.13970
P-value: 0.25828
Fail to reject the null hypothesis: Data follows the uniform distribution.
```

Lab 5: Test random number using Chi-square method

```
import numpy as np

from scipy import stats

# Generate random numbers from a uniform distribution

sample_size = 100  # Number of random numbers

bins = 10  # Number of bins for chi-square test

data = np.random.uniform(0, 1, sample_size)  # Generate random numbers

# Generate data

observed_frequencies, bin_edges = np.histogram(data, bins=bins)

# Expected frequencies assuming uniform distribution

expected_frequencies = np.full(bins, sample_size / bins)

# Display generated random numbers and frequencies

print("Generated Random Numbers:")

print(data)

print("\nObserved Frequencies:")

print(observed_frequencies)

# Perform Chi-Square test

chi_square_statistic, p_value = stats.chisquare(observed_frequencies, expected_frequencies)

# Output the results

print(f"\nChi-Square Statistic: {chi_square_statistic}")

print(f"P-value: {p_value}")

# Interpretation of results

alpha = 0.05  # Significance level

if p_value > alpha:

print("Fail to reject the null hypothesis: Data follows the uniform distribution.")

else:

print("Reject the null hypothesis: Data does not follow the uniform distribution.")
```

Output:

```
Generated Random Numbers:
[0.4249135  0.7426253  0.5669347  0.95982972 0.30976505 0.65230306
 0.81792652 0.53249557 0.36550714 0.98738834 0.39273051 0.21146228
 0.78509545 0.45858242 0.02123066 0.02104144 0.52980326 0.94128846
 0.9801884  0.10001276 0.25622829 0.73424042 0.48127754 0.7409195
 0.62761415 0.09210159 0.16466175 0.95304277 0.35009519 0.66552006
 0.79899082 0.90854464 0.65005808 0.00835549 0.99671019 0.57249848
 0.86718741 0.78701024 0.68880003 0.78037198 0.94348097 0.4131254
 0.82890019 0.19300527 0.86000888 0.28509672 0.59097834 0.97973274
 0.33934382 0.93530017 0.52666967 0.79774987 0.11247673 0.60895774
 0.86593754 0.6235162  0.35696099 0.87931285 0.58972937 0.81694309
 0.55957318 0.48955827 0.71349532 0.40833893 0.77840955 0.67046266
 0.41261181 0.92767327 0.86007893 0.26113889 0.41587176 0.4085359
 0.24110842 0.6395246  0.48932428 0.5857891  0.17654522 0.35453009
 0.57774208 0.72437676 0.99942335 0.57442245 0.36081385 0.06601217
 0.09523378 0.73596433 0.19007802 0.20976553 0.83925286 0.61733039
 0.04145373 0.24497604 0.27582585 0.58481681 0.46592074 0.27477874
 0.09848653 0.03995395 0.60202427 0.27429198]

Observed Frequencies:
[10  5 10  8 11 13 10 12  9 12]

Chi-Square Statistic: 4.8
P-value: 0.8513825753567951
Fail to reject the null hypothesis: Data follows the uniform distribution.
```

Lab 6: Test random number using Poker Test

```
import numpy as np
from collections import Counter
from scipy import stats


# Function to categorize numbers into poker hands
def categorize_hand(hand):
    counts = Counter(hand)
    unique_counts = sorted(counts.values(), reverse=True)


    if unique_counts == [5]:  # Five of a kind
        return "Five of a kind"
    elif unique_counts == [4, 1]:  # Four of a kind
        return "Four of a kind"
    elif unique_counts == [3, 2]:  # Full house
        return "Full house"
    elif unique_counts == [3, 1, 1]:  # Three of a kind
        return "Three of a kind"
    elif unique_counts == [2, 2, 1]:  # Two pair
        return "Two pair"
    elif unique_counts == [2, 1, 1, 1]:  # One pair
        return "One pair"
    else:  # All different digits
        return "All different"


# Generate random hands (each hand has 5 digits from 0-9)
sample_size = 100  # Number of random hands
hand_size = 5  # Number of digits in each hand
```

```
data = np.random.randint(0, 10, (sample_size, hand_size))

# Display generated hands

print("Generated Poker Hands (First 10 shown):")

for hand in data[:10]:

  print(hand)

# Categorize each hand

hand_categories = [categorize_hand(hand) for hand in data]

# Count occurrences of each category

category_counts = Counter(hand_categories)

# Expected probabilities for uniform distribution

expected_probabilities = {

"Five of a kind": 0.0001,

"Four of a kind": 0.0045,

"Full house": 0.009,

"Three of a kind": 0.072,

"Two pair": 0.108,

"One pair": 0.504,

"All different": 0.3024

}

# Expected frequencies based on probabilities

expected_frequencies = {k: v * sample_size for k, v in expected_probabilities.items()}

# Perform Chi-Square test

observed = [category_counts.get(k, 0) for k in expected_probabilities.keys()]

expected = [expected_frequencies[k] for k in expected_probabilities.keys()]

chi_square_statistic, p_value = stats.chisquare(observed, expected)

# Output observed hand categories

print("\nObserved Hand Categories:")

for category, count in category_counts.items():

  print(f"{category}: {count}")
```

```
  print(f"\nChi-Square Statistic: {chi_square_statistic}")
  print(f"P-value: {p_value}")
# Interpretation of results
alpha = 0.05  # Significance level
if p_value > alpha:
  print("Fail to reject the null hypothesis: Data follows expected poker hand distribution.")
else:
  print("Reject the null hypothesis: Data does not follow expected poker hand distribution.")
```

Output:

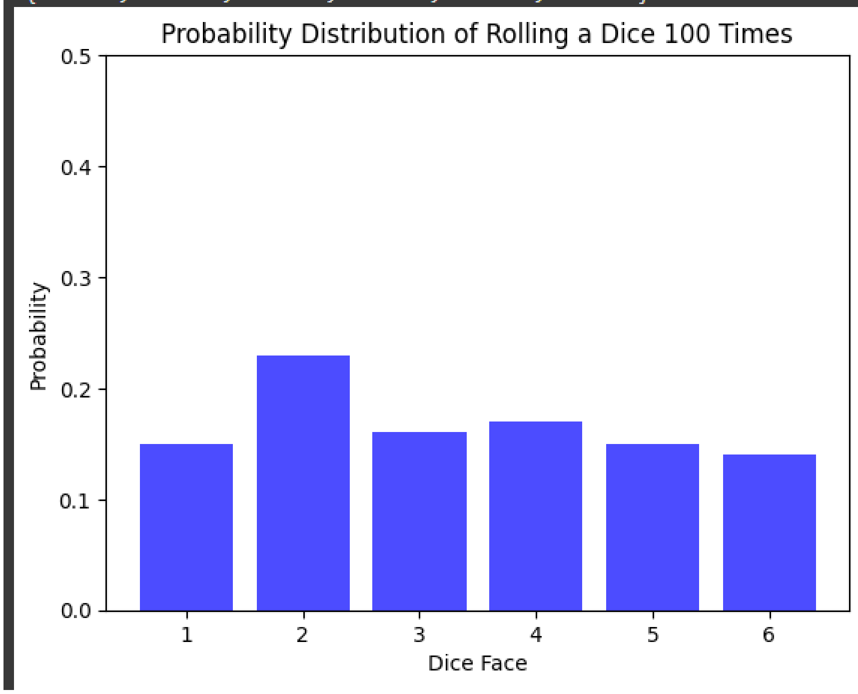
```
Generated Poker Hands (First 10 shown):
[6 0 5 9 6]
[2 3 5 5 9]
[2 1 4 4 5]
[4 3 1 0 5]
[4 0 3 1 8]
[9 1 6 3 0]
[2 6 8 2 8]
[4 7 8 3 9]
[0 7 6 7 3]
[4 4 1 0 9]

Observed Hand Categories:
One pair: 45

Chi-Square Statistic: 2.2552910052910042
P-value: 0.8947865710811523
All different: 33

Chi-Square Statistic: 2.2552910052910042
P-value: 0.8947865710811523
Two pair: 14

Chi-Square Statistic: 2.2552910052910042
P-value: 0.8947865710811523
Three of a kind: 7

Chi-Square Statistic: 2.2552910052910042
P-value: 0.8947865710811523
Full house: 1

Chi-Square Statistic: 2.2552910052910042
P-value: 0.8947865710811523
Fail to reject the null hypothesis: Data follows expected poker hand distribution.
```

Lab 7: WAP for random number generation in dice roll. If I roll dice 100,1000,10000 times, what is the result? (Probability distribution)

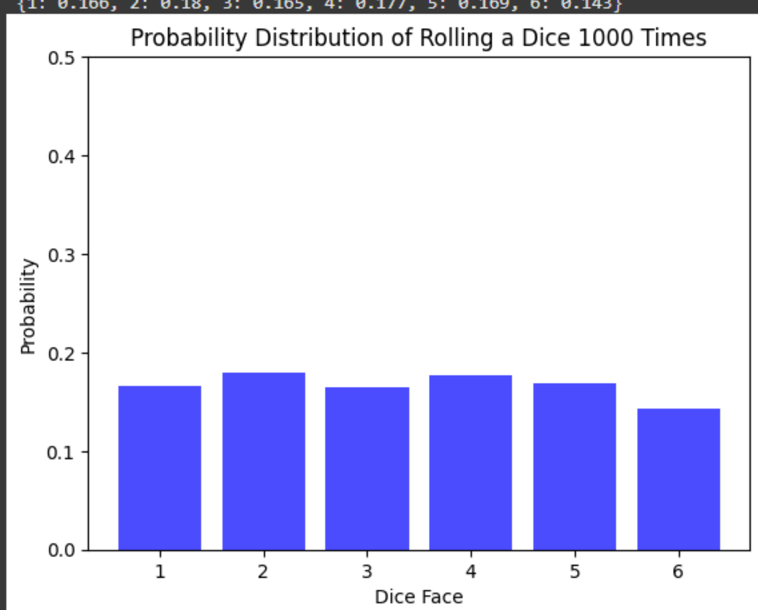
```
import random
import matplotlib.pyplot as plt


def roll_dice(num_rolls):
    results = [random.randint(1, 6) for _ in range(num_rolls)]
    frequencies = {i: results.count(i) / num_rolls for i in range(1, 7)}
    return frequencies


def plot_distribution(distribution, num_rolls):
    faces = list(distribution.keys())
    probabilities = list(distribution.values())


    plt.bar(faces, probabilities, tick_label=faces, color='blue', alpha=0.7)
    plt.xlabel("Dice Face")
    plt.ylabel("Probability")
    plt.title(f"Probability Distribution of Rolling a Dice {num_rolls} Times")
    plt.ylim(0, 0.5)
    plt.show()


# Run simulations and plot results
for rolls in [100, 1000, 10000]:
    distribution = roll_dice(rolls)
    print(f"Probability distribution for {rolls} rolls: \n", distribution)
    plot_distribution(distribution, rolls)
```

Output:

```
Probability distribution for 100 rolls:
 {1: 0.15, 2: 0.23, 3: 0.16, 4: 0.17, 5: 0.15, 6: 0.14}
```




```
Probability distribution for 1000 rolls:
 {1: 0.166, 2: 0.18, 3: 0.165, 4: 0.177, 5: 0.169, 6: 0.143}
```




```
Probability distribution for 10000 rolls:
```

```
Probability distribution for 10000 rolls:
 {1: 0.1633, 2: 0.1694, 3: 0.1691, 4: 0.1685, 5: 0.158, 6: 0.1717}
```