Open in Colab

chapter 05 Sets and probability

In [ ]:
```
from sympy import FiniteSet #construction of set
s = FiniteSet(2,4,6)
s
```

Out[ ]: $\{2, 4, 6\}$

In [ ]:
```
from sympy import FiniteSet
from fractions import Fraction
s = FiniteSet(1, 1.5, Fraction(1,5))
s
```

Out[ ]: $\left\{\frac{1}{5}, 1, 1.5\right\}$

In [ ]:
```
s = FiniteSet(1,1.5,3)
len(s)
```

Out[ ]: 3

In [ ]:
```
4 in s #checking the number is wheather in set or not
```

Out[ ]: False

In [ ]:
```
s = FiniteSet()
s
```

Out[ ]: $\emptyset$

In [ ]:
```
members = [1,2,3]
s = FiniteSet(*members)   #creating sets from list or tuples
s
```

Out[ ]: $\{1, 2, 3\}$

In [ ]:
```
from sympy import FiniteSet
members = [1,2,3,2]
FiniteSet(*members)
```

Out[ ]: $\{1, 2, 3\}$

In [ ]:
```
from sympy import FiniteSet
s = FiniteSet(1,2,3)
```

```
for member in s:
  print(member)
```

```
1
2
3
```

In [ ]:
```
from sympy import FiniteSet
s = FiniteSet(3,4,5)
t = FiniteSet(5,4,3)
s == t
```

Out[ ]: True

Subsets, Supersets, and Power Sets

In [ ]:
```
s = FiniteSet(1)
t = FiniteSet(1,2)
s.is_subset(t)
```

Out[ ]: True

In [ ]:
```
t.is_subset(s)
```

Out[ ]: False

In [ ]:
```
print(s.is_subset(t))
print(t.is_subset(s))
```

```
True
False
```

In [ ]:
```
s = FiniteSet(1,2,3)
ps = s.powerset()
ps
```

Out[ ]: $\{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$

In [ ]:
```
len(ps)
```

Out[ ]: 8

checking the relationships of proper superset & powerset

In [ ]:
```
from sympy import FiniteSet
s = FiniteSet(1,2,3)
t = FiniteSet(1,2,3)
s.is_proper_subset(t)
```

Out[ ]: False

In [ ]:
```
t.is_proper_superset(s)
```

Out[ ]: False

In [ ]:
```
t = FiniteSet(1,2,3,4)
s.is_proper_subset(t)
```

Out[ ]: True

In [ ]:
```
t.is_proper_superset(s)
```

Out[ ]: True

union & intersection

In [ ]:
```
from sympy import FiniteSet
s = FiniteSet(1,2,3)
t = FiniteSet(2,4,6)
s.union(t)
```

Out[ ]: $\{1, 2, 3, 4, 6\}$

In [ ]:
```
s = FiniteSet(1,2)
t = FiniteSet(2,3)
s.intersection(t)
```

Out[ ]: $\{2\}$

In [ ]:
```
from sympy import FiniteSet
s = FiniteSet(1,2,3)
t = FiniteSet(2,4,6)
u = FiniteSet(3,5,7)
s.union(t).union(u)
```

Out[ ]: $\{1, 2, 3, 4, 5, 6, 7\}$

In [ ]:
```
s.intersect(t).intersect(u)
```

Out[ ]: $\emptyset$

cartesian product

In [ ]:
```
from sympy import FiniteSet
s = FiniteSet(1,2)
t = FiniteSet(3,4)
p = s*t
p
```

Out[ ]: $\{1,2\} \times \{3,4\}$

In [ ]:
```
for elem in p:
  print(elem)
```

```
(1, 3)
(2, 3)
(1, 4)
(2, 4)
```

In [ ]:
```
len(p) == len(s)*len(t)
```

Out[ ]: True

In [ ]:
```
from sympy import FiniteSet
s = FiniteSet(1,2)
p = s**3
p
```

Out[ ]: $\{1,2\}^3$

In [ ]:
```
for elem in p:

  print(elem)
```

```
(1, 1, 1)
(2, 1, 1)
(1, 2, 1)
(2, 2, 1)
(1, 1, 2)
(2, 1, 2)
(1, 2, 2)
(2, 2, 2)
```

applying the formula to multiple sets of variables

In [ ]:
```
from sympy import FiniteSet, pi

def time_period(length):
  g = 9.8
  T = 2*pi*(length/g)**0.5
  return T


if __name__ == '__main__':
  L=FiniteSet(15,18,21,22.5,25)
  for l in L:
    t = time_period(1/100)
    print('length: {0} cm time period: {1:3f} s'.format(float(1), float(t)))
```

```
length: 1.0 cm time period: 0.200709 s
length: 1.0 cm time period: 0.200709 s
length: 1.0 cm time period: 0.200709 s
length: 1.0 cm time period: 0.200709 s
length: 1.0 cm time period: 0.200709 s
```

Different Gravity, Different Results

In [ ]:
```
from sympy import FiniteSet, pi

def time_period(length,g):
  T = 2*pi*(length/g)**0.5
  return T


if __name__ == '__main__':

  L = FiniteSet(15,18,21,22.5,25)
  g_values = FiniteSet(9.8, 9.78,9.83)
  print('{0:^15}{1:^15}{2:^15}'.format('Length(cm)','Gravity(m/s^2)','Time pe
  for elem in L*g_values:
    l = elem[0]
    g = elem[1]
    t = time_period(1/100,g)
    print('{0:^15}{1:^15}{2:^15.3}'.format(float(l), float(g), float(t)))
```

```
Length(cm)   Gravity(m/s^2)   Time period
   22.5           9.78           0.201
   15.0           9.78           0.201
   22.5            9.8           0.201
   18.0           9.78           0.201
   15.0            9.8           0.201
   22.5           9.83            0.2
   21.0           9.78           0.201
   18.0            9.8           0.201
   15.0           9.83            0.2
   25.0           9.78           0.201
   21.0            9.8           0.201
   18.0           9.83            0.2
   25.0            9.8           0.201
   21.0           9.83            0.2
   25.0           9.83            0.2
```

probability

In [ ]:
```
def probability(space, event):
  return len(event)/len(space)
```

In [ ]:
```
def probability(space,event):
  return len(event)/len(space)


def check_prime(number):
  if number != 1:
    for factor in range(2,number):
      if number % factor == 0:
        return False
  else:
        return False
  return True
```

```
if __name__ =='__main__':

  space = FiniteSet(*range(1,21))
  primes =[]
  for num in s:
    if check_prime(num):
      primes.append(num)
  event = FiniteSet(*primes)
  p = probability(space,event)

  print('Sample space:{0}'.format(space))
  print('event:{0}'.format(event))
  print('probability of rolling a prime:{0:5f}'.format(p))
```

```
Sample space:{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 1
9, 20}
event:{2}
probability of rolling a prime:0.050000
```

In [ ]:
```
if __name__ == '__main__':
  space = range(1,21)
  primes = []
  for num in space:
    if check_prime(num):
      primes.append(num)
  p = probability(space, prime)
```

Probability of Event A or Event B

In [42]:
```
from sympy import FiniteSet
s = FiniteSet(1,2,3,4,5,6)
a = FiniteSet(2,3,5)
b = FiniteSet(1,3,5)
e = a.union(b)
len(e)/len(s)
```

Out[42]: 0.6666666666666666

In [45]:
```
from sympy import FiniteSet
s = FiniteSet(1,2,3,4,5,6)
a = FiniteSet(2,3,5)
b = FiniteSet(1,3,5)
e = a.intersect(b)
len(e)/len(s)
```

Out[45]: 0.3333333333333333

Generating random numbers

In [47]:
```
import random #simulating a die roll
random.randint(1,6)
```

Out[47]: 2

In [53]:
```
random.randint(1,6)
```

Out[53]: 3

Roll a die until the total score is 20

In [56]:
```
import matplotlib.pyplot as plt
import random

target_score = 20
def roll():
  return random.randint(1,6)

if __name__ == '__main__':
  score = 0
  num_rolls = 0
  while score < target_score:
    die_roll = roll()
    num_rolls +=1
    print('rolled:{0}'.format(die_roll))
    score += die_roll

    print('score of {0} reached in {1} rolls '.format(score, num_rolls))
```

```
rolled:6
score of 6 reached in 1 rolls
rolled:4
score of 10 reached in 2 rolls
rolled:1
score of 11 reached in 3 rolls
rolled:2
score of 13 reached in 4 rolls
rolled:1
score of 14 reached in 5 rolls
rolled:3
score of 17 reached in 6 rolls
rolled:3
score of 20 reached in 7 rolls
```

In [58]:
```
from sympy import FiniteSet
import random


def find_prob(target_score, max_rolls):

  die_sides = FiniteSet(1,2,3,4,5,6)
  #sample space
  s = die_sides**max_rolls
  #find the event set
  if max_rolls > 1:
    success_rolls = []
    for elem in s :
      if sum(elem) >=target score:
```

```
        success_rolls.append(elem)
    else:
      if target_score > 6 :
        success_rolls = []
      else:
        success_rolls = []
        for roll in die_sides:
          if roll >= target_score:
            success_rolls.append(roll)

    e = FiniteSet(*success_rolls)
    #calculating the probability of reaching target score
    return len(e)/len(s)


if __name__ == '__main__':


    target_score = int(input('enter the target score:'))
    max_rolls = int(input('enter the maximum number of the rolls allowed:'))
    p = find_prob(target_score, max_rolls)
    print('probability:{0:.5f}'.format(p))
```

```
enter the target score:25
enter the maximum number of the rolls allowed:5
probability:0.03241
```

non uniform random numbers

In [61]:

```
import random
def toss():
  #0 -> heads, 1 - > tails
  if random.random() <2/3:
    return 0

  else:
    return 1

    # Call the toss function and print the result
result = toss()
print("Result of the toss:", "Heads" if result == 0 else "Tails")
```

```
Result of the toss: Heads
```

In [72]:

```
import random

def get_index(probability):
  c_probability = 0
  sum_probability = []
  for p in probability:
    c_probability += p
    sum_probability.append(c_probability)

  r = random.random()
  for index, sp in enumerate(sum_probability):
    if r <= sp:
      return index
```

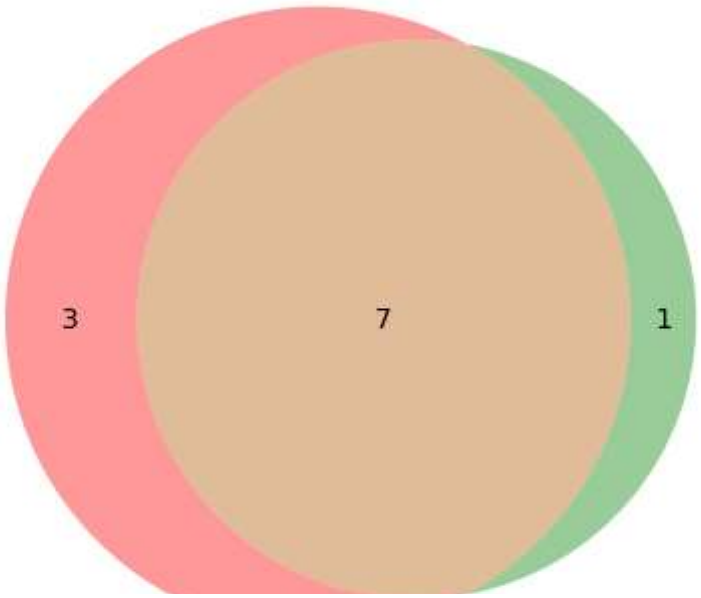
```
  return len(probability)-1
def dispense():
  bills = [5,10,20,50]
  probability = [1/6, 1/6, 1/3, 2/3]
  bill_index = get_index(probability)
  return bills[bill_index]

  # Example usage
if __name__ == '__main__':
  for _ in range(10):
      # Dispense 10 bills
      print("Dispensed bill:", dispense())
```

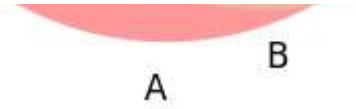
```
Dispensed bill: 10
Dispensed bill: 5
Dispensed bill: 20
Dispensed bill: 20
Dispensed bill: 20
Dispensed bill: 50
Dispensed bill: 20
Dispensed bill: 5
Dispensed bill: 5
Dispensed bill: 50
```

Programming challenges

In [74]:

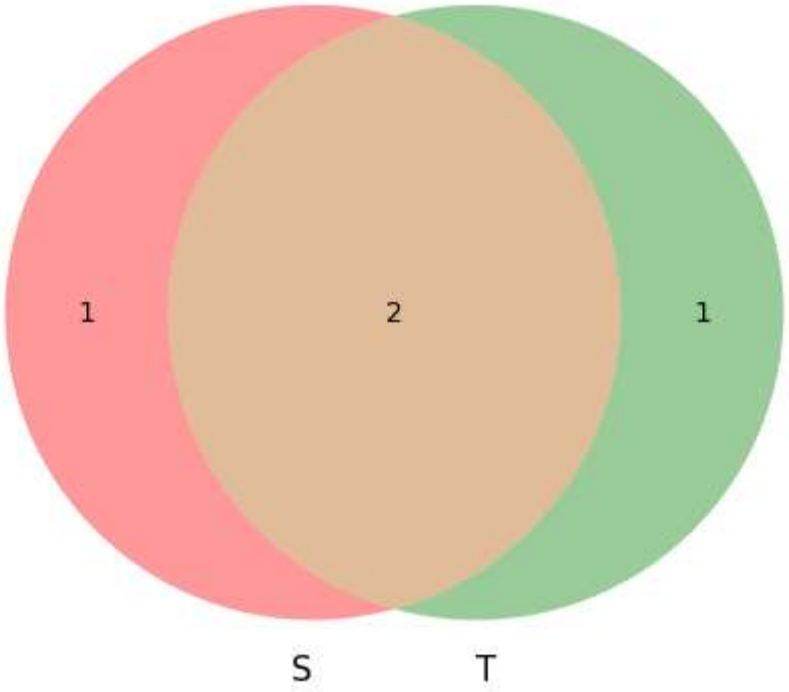
```
from matplotlib_venn import venn2
import matplotlib.pyplot as plt
from sympy import FiniteSet
def draw_venn(sets):
    venn2(subsets=sets)
    plt.show()

if __name__ == '__main__':
    s1 = FiniteSet(1, 3, 5, 7, 9, 11, 13, 15, 17, 19)
    s2 = FiniteSet(2, 3, 5, 7, 11, 13, 17, 19)
    draw_venn([s1, s2])
```





In [75]:
```
venn2(subsets=(a,b), set_labels=('S', 'T'))
```

Out[75]: <matplotlib_venn._common.VennDiagram at 0x786dee11dc90>



In [76]:
```
import csv

# Sample data (replace with actual data)
data = [
    [1, 1, 0],
    [2, 1, 1],
    [3, 0, 1],
    # Add more student data here
]

# Write data to CSV file
with open('sports.csv', 'w', newline='') as csvfile:
    fieldnames = ['StudentID', 'Football', 'Others']
    writer = csv.writer(csvfile)
    writer.writerow(fieldnames)
    writer.writerows(data)

print("CSV file 'sports.csv' created successfully!")
```

CSV file 'sports.csv' created successfully!

In [78]:
```
import pandas as pd
df = pd.read_csv('sports.csv')
df
```

Out[78]:

	StudentID	Football	Others
0	1	1	0
1	2	1	1
2	3	0	1

2: Law of Large Numbers

In [79]:

```
e = 1*(1/6) + 2*(1/6) + 3*(1/6) + 4*(1/6) + 5*(1/6) + 6*(1/6)
e
```

Out[79]: 3.5

3: How Many Tosses Before You Run Out of Money?

In [86]:

```
 import random

def roll_die(num_trials):

    results = []
    for _ in range(num_trials):
        result = random.randint(1, 6)
        results.append(result)
    return results

def calculate_average(results):

   # Calculates the average value of a list of results.

         #results: A list of numbers.

    total = sum(results)
    average = total / len(results)
    return average

if __name__ == "__main__":
    expected_value = 3.5
    num_trials_list = [100, 1000, 10000, 100000, 500000]

    print(f"Expected value: {expected_value}")

    for num_trials in num_trials_list:
        results = roll_die(num_trials)
        trial_average = calculate_average(results)
        print(f"Trials: {num_trials} Trial average: {trial_average}")
```

```
Expected value: 3.5
```

```
Trials: 100 Trial average: 3.39
Trials: 1000 Trial average: 3.472
Trials: 10000 Trial average: 3.5005
Trials: 100000 Trial average: 3.49304
Trials: 500000 Trial average: 3.497142
```

In [85]:
```python
import random # Import the random module

def coin_toss_game(starting_amount): # Simulate the coin toss game
    current_amount = starting_amount # Initialize current amount
    num_tosses = 0 # Initialize number of tosses

    while current_amount > 0: # Continue while player has money
        toss = random.choice(["Heads", "Tails"]) # Choose heads or tails rand
        current_amount += 1 if toss == "Heads" else -1.5 # Update amount base
        num_tosses += 1 # Increment number of tosses

    return num_tosses, current_amount # Return num_tosses and current_amount

if __name__ == "__main__": # Main execution block
    starting_amount = float(input("Enter your starting amount: ")) # Get star
    num_tosses, current_amount = coin_toss_game(starting_amount) # Simulate g
    print(f"Game over! (Current amount: ${current_amount:.2f}, Coin tosses: {
```

```
Enter your starting amount: 10
Game over! (Current amount: $0.00, Coin tosses: 75)
```

4: Shuffling a Deck of Cards

In [87]:
```python
import random # Import the random module

def shuffle_deck(): # Shuffles a deck of 52 cards
    deck = list(range(1, 53)) # Create a list of integers 1 to 52
    random.shuffle(deck) # Shuffle the list in-place
    return deck # Return the shuffled list

if __name__ == "__main__": # Main execution block
    shuffled_deck = shuffle_deck() # Shuffle the deck
    print(shuffled_deck) # Print the shuffled deck
```

```
[37, 11, 17, 46, 34, 21, 7, 9, 43, 51, 25, 14, 44, 33, 26, 28, 23, 27, 4, 20, 4
7, 1, 50, 48, 5, 22, 40, 18, 38, 16, 13, 19, 30, 2, 45, 52, 29, 41, 8, 36, 10,
35, 32, 3, 31, 15, 39, 6, 42, 24, 12, 49]
```

In [88]:
```python
import random
x = [1,2,3,4]
random.shuffle(x)
x
```

Out[88]: [4, 3, 1, 2]

In [89]:
```python
import random
```

```
class Card:
    def __init__(self, suit, rank):
        self.suit = suit
        self.rank = rank

def create_deck():
    suits = ['spades', 'clubs', 'hearts', 'diamonds']
    ranks = ['ace', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'jack', 'qu
    deck = []
    for suit in suits:
        for rank in ranks:
            deck.append(Card(suit, rank))
    return deck

def shuffle_deck(deck):
    random.shuffle(deck)
    return deck

if __name__ == "__main__":
    deck = create_deck()
    shuffled_deck = shuffle_deck(deck)

    for card in shuffled_deck:
        print(f"{card.rank} of {card.suit}")
```

```
6 of spades
jack of clubs
9 of spades
10 of diamonds
8 of spades
10 of clubs
queen of clubs
5 of diamonds
8 of hearts
jack of hearts
jack of spades
4 of clubs
8 of clubs
king of spades
7 of hearts
9 of hearts
4 of hearts
queen of hearts
5 of spades
9 of diamonds
7 of diamonds
7 of clubs
king of clubs
ace of clubs
3 of spades
queen of diamonds
4 of spades
6 of diamonds
2 of diamonds
ace of diamonds
4 of diamonds
9 of clubs
3 of hearts
```

```
3 of diamonds
2 of clubs
6 of clubs
7 of spades
king of diamonds
10 of spades
2 of spades
10 of hearts
king of hearts
5 of clubs
5 of hearts
jack of diamonds
queen of spades
8 of diamonds
3 of clubs
ace of spades
6 of hearts
2 of hearts
ace of hearts
```

5: Estimating the Area of a Circle

In [90]:

```python
import random
import math

def estimate_circle_area(radius, num_darts):
    """
    Estimates the area of a circle using the Monte Carlo method.

    Args:
        radius: The radius of the circle.
        num_darts: The number of darts to throw.

    Returns:
        The estimated area of the circle.
    """

    num_hits = 0
    side = 2 * radius  # Side of the square

    for _ in range(num_darts):
        # Generate random coordinates within the square
        x = random.uniform(0, side)
        y = random.uniform(0, side)

        # Check if the dart lands within the circle
        if (x - radius) ** 2 + (y - radius) ** 2 <= radius ** 2:
            num_hits += 1

    # Calculate the estimated area
    estimated_area = (num_hits / num_darts) * side ** 2

    return estimated_area

if __name__ == "__main__":
    radius = 2
```

```
    actual_area = math.pi * radius ** 2

    num_darts_list = [1000, 100000, 1000000]

    print(f"Radius: {radius}")
    print(f"Area: {actual_area}")

    for num_darts in num_darts_list:
        estimated_area = estimate_circle_area(radius, num_darts)
        print(f"Estimated ({num_darts} darts): {estimated_area}")
```

```
Radius: 2
Area: 12.566370614359172
Estimated (1000 darts): 12.528
Estimated (100000 darts): 12.58144
Estimated (1000000 darts): 12.571456
```

Estimating the Value of Pi

In [91]:

```
import random

def estimate_pi(num_darts):

    num_hits = 0

    for _ in range(num_darts):
        # Generate random coordinates within the unit square
        x = random.uniform(-1, 1)
        y = random.uniform(-1, 1)

        # Check if the dart lands within the unit circle
        if x**2 + y**2 <= 1:
            num_hits += 1

    # Calculate the estimated value of pi
    pi_estimate = 4 * num_hits / num_darts

    return pi_estimate

if __name__ == "__main__":
    num_darts_list = [1000, 10000, 100000, 1000000]

    for num_darts in num_darts_list:
        estimated_pi = estimate_pi(num_darts)
        print(f"Estimated pi with {num_darts} darts: {estimated_pi}")
```

```
Estimated pi with 1000 darts: 3.128
```