Lab 9

July 19, 2024

Dylan Liesenfelt

1 Lab 9 - Counting and Probability

1.1 Exercise 1: Menu Choices

A restaurant offers 4 choices of appetizers, 5 choices of main courses, and 3 choices of desserts. How many different meals can be ordered if each meal includes one appetizer, one main course, and one dessert?

1.1.1 Solution:

Use the rule of products. Total number of meals = 4 appetizers * 5 main courses * 3 desserts = 60 meals.

[ ]:
```
# Rule of Products Simulation for Exercise 1
appetizers = 9          # TASK: Please change the choices: Okay
main_courses = 5
desserts = 4
total_meals = appetizers * main_courses * desserts
print(f"Total number of different meals: {total_meals}")
```

```
Total number of different meals: 180
```

1.2 Exercise 2: Password Creation

A website requires a password that consists of 3 letters followed by 3 digits. How many unique passwords can be created if letters and digits can be repeated?

1.2.1 Solution:

There are 26 possibilities for each letter and 10 possibilities for each digit. Total number of passwords = 26^3 * 10^3.

[ ]:
```
# Rule of Products Simulation for Exercise 2
letters = 26 ** 9                      # TASK: Please change the numbers: Okay
digits = 10 ** 7
total_passwords = letters * digits
print(f"Total number of unique passwords: {total_passwords}")
```

```
Total number of unique passwords: 54295036789760000000
```

1.3 Exercise 3: Book Arrangement

How many ways can 5 different books be arranged on a shelf?

1.3.1 Solution

This is a permutation problem with all 5 books being distinct. Total arrangements = 5! (factorial of 5).

```
[ ]: # Permutations Simulation

# Solution 1
total_arrangements = 1
for i in range(1, 20):            # TASK: Please change the numbers: Okay
    total_arrangements *= i
print(f"Total arrangements of books: {total_arrangements}")

# Solution 2
def generate_permutations(sequence, items):
    if len(items) == 0:
        all_permutations.append(sequence)
    else:
        for i in range(len(items)):
            # Generate a new sequence that includes the current item
            new_sequence = sequence + [items[i]]
            # Create a new list of items without the current item
            remaining_items = items[:i] + items[i+1:]
            # Recursively generate permutations with the updated sequence and␣
↪remaining items
            generate_permutations(new_sequence, remaining_items)

# Initialize the list of all permutations
all_permutations = []

# Define the books
books = ['Q', 'R', 'S', 'T', 'U']            # TASK: Try to modify the list:␣
↪Okay

# Start generating permutations with an empty sequence and the full list of␣
↪items
generate_permutations([], books)

# The total number of arrangements is the length of the list of all permutations
total_arrangements = len(all_permutations)

print(f"Total arrangements of books: {total_arrangements}")
```

```
# This print statement is to indicate the process, in practice, you'd probably␣
↪want to do more with the permutations.
```

```
Total arrangements of books: 121645100408832000
Total arrangements of books: 120
```

1.4 Exercise 4: Committee Selection

A committee of 3 people is to be formed from a group of 8 candidates. How many different committees are possible?

1.4.1 Solution

We're dealing with the concept of combinations, since the order in which committee members are selected does not matter. $C(8,3) = \frac{8!}{3!(8-3)!} = \frac{8\times7\times6}{3\times2\times1} = \frac{40320}{6\times120} = \frac{40320}{720} = 56$

[ ]:
```
# Combinations Simulation for Exercise 2

# SOLUTION 1 - Applying the formula
def factorial(n):
    """Calculate the factorial of n."""
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result

def combinations(n, k):
    """Calculate the number of combinations (n choose k)."""
    return factorial(n) // (factorial(k) * factorial(n - k))

# Calculate the number of ways to form a committee of 3 from 8 candidates
total_committees_formula = combinations(2, 9)                              #␣
↪TASK: Change the numbers: Okay
print(f"Total committees (using formula): {total_committees_formula}")

# SOLUTION 2 - Doing a simulation
def generate_combinations(candidates, k, start=0, path=[], result=[]):
    """Generate all combinations of k candidates from the list."""
    # If the path length is k, append it to the result
    if len(path) == k:
        result.append(path)
        return
    for i in range(start, len(candidates)):
        # Generate combinations by including the current candidate and moving␣
↪to the next
        generate_combinations(candidates, k, i + 1, path + [candidates[i]],␣
↪result)
```

```
candidates = list(range(8))  # Representing candidates as numbers 0 through 7  ␣
↪ # TASK: Change the numbers
combinations_result = []
generate_combinations(candidates, 3, result=combinations_result)

# The number of ways to form the committees is the length of the␣
↪combinations_result list
total_committees_simulation = len(combinations_result)
print(f"Total committees (via simulation): {total_committees_simulation}")
```

```
Total committees (using formula): 0
Total committees (via simulation): 56
```

1.5 Exercise 5: Soccer Team

How many ways can 11 players be selected from a squad of 16 to start a soccer match?

1.5.1 Solution

This is a combination problem since the order does not matter. Total ways = C(16, 11) = 16! / (11! * (16-11)!).

[ ]:
```
# Solution 1: Applying the Combinations Formula
def factorial(n):
    """Calculate the factorial of n."""
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result

def combinations(n, k):
    """Calculate the number of combinations (n choose k)."""
    return factorial(n) // (factorial(k) * factorial(n - k))

# Calculate the number of ways to select 11 players from a squad of 16
total_selections_formula = combinations(55, 11)                              ␣
↪         # TASK: Change the numbers:Okay
print(f"Total ways to select 11 players (using formula):␣
↪{total_selections_formula}")

#Solution 2: Simulating Player Selection
def generate_combinations(candidates, k, start=0, path=[], result=[]):
    """Generate all combinations of k candidates from the list."""
    # If the path length is k, append it to the result
    if len(path) == k:
        result.append(path)
        return
    for i in range(start, len(candidates)):
```

```
        # Generate combinations by including the current candidate and moving
 ↪to the next
        generate_combinations(candidates, k, i + 1, path + [candidates[i]],
 ↪result)

candidates = list(range(11))  # Representing players as numbers 0 through 15
 ↪     # TASK change the numbers: Okay
combinations_result = []
generate_combinations(candidates, 11, result=combinations_result)

# The number of ways to select the players is the length of the
 ↪combinations_result list
total_selections_simulation = len(combinations_result)
print(f"Total ways to select 11 players (via simulation):
 ↪{total_selections_simulation}")
```

```
Total ways to select 11 players (using formula): 119653565850
Total ways to select 11 players (via simulation): 1
```

1.6 Exercise 5: Birthday Problem

Given a room with n people, what is the probability that at least two people share the same birthday? Assume a year of 365 days, and ignore leap years.

1.6.1 Solution

The probability that all n people have unique birthdays is calculated by considering the probability of each successive person having a birthday different from those previously considered. For the first person, any birthday is acceptable, giving a probability of $\frac{365}{365}$. The second person must have a different birthday than the first, giving a probability of $\frac{364}{365}$, and so on, until the nth person, who must have a birthday different from the previous $n-1$ people, giving a probability of $\frac{365-n+1}{365}$.

Therefore, the probability $P_{\text{unique}}(n)$ of n people all having unique birthdays is the product of these probabilities:

$P_{\text{unique}}(n) = \frac{365}{365} \times \frac{364}{365} \times \cdots \times \frac{365-n+1}{365}$

The probability that at least two people in a group of n share the same birthday is the complement of $P_{\text{unique}}(n)$:

$P(n) = 1 - P_{\text{unique}}(n)$

For $n = 23$, this becomes:

$P(23) = 1 - \left(\frac{365}{365} \times \frac{364}{365} \times \cdots \times \frac{343}{365}\right)$

$P(23) \approx 0.5073$

Thus, the probability that in a group of 23 people, at least two share the same birthday is approximately 50.73%.

$P(n) = 1 - \prod_{i=0}^{n-1} \frac{365-i}{365}$ This formula represents the complement of the probability that all n people have different birthdays, with the product running from i=0 to n-1.

[ ]:
```python
import random

# SOLUTION 1 - Applying the formula
def theoretical_birthday_probability(n):
    probability_unique = 1.0
    for i in range(n):
        probability_unique *= (365 - i) / 365
    return 1 - probability_unique

# Example: Probability for n = 23
n = 23          # TASK: Change the numbers
print(f"Theoretical probability for {n} people: 
 ↪{theoretical_birthday_probability(n)}")

# SOLUTION 2 - Doing a simulation
def simulate_birthday_problem(n, simulations=505050):                # TASK: 
 ↪Change the numbers: Okay
    shared_birthday_count = 0
    for _ in range(simulations):
        # Generate random birthdays for n participants
        birthdays = [random.randint(1, 365) for _ in range(n)]
        # Check if there is at least one shared birthday
        if len(birthdays) != len(set(birthdays)):
            shared_birthday_count += 1
    # Calculate probability based on simulation
    return shared_birthday_count / simulations

# Example: Simulation for n = 23
n = 17                                                                  # TASK: 
 ↪Change the numbers
simulated_probability = simulate_birthday_problem(n)
print(f"Simulated probability for {n} people: {simulated_probability}")
```

```
Theoretical probability for 23 people: 0.5072972343239857
Simulated probability for 23 people: 0.525
```