Task 1: ***Financial application - initial deposit amount.***
Suppose you want to deposit a certain amount of money into a savings account with a fixed annual interest rate. Write a program that prompts the user to enter the final account value, the annual interest rate in percent, and the number of years, and then displays the initial deposit amount. The initial deposit amount can be obtained using the following formula:

$$initialDepositAmount = \frac{finalAccountValue}{(1 + monthlyInterestRate)^{\{numberOfMonths\}}}$$

**Here is a sample run:**

```
<output>

Enter final account value: 1000

Enter annual interest rate in percent: 4.25

Enter number of years: 5

Initial deposit value is 808.8639197424636

<end output>
```

*Task 2:* ***Future tuition.***

Print a table that prompts the user to enter the current tuition and displays a table that displays how many years the tuition will be doubled if the tuition is increased by 3%, 4%, and up to 10% annually.

**Here is a sample run:**

```
<output>

Enter the current year tuition: 15000

     Rate     Number of Years

     3.0%                  24

     4.0%                  18

     5.0%                  15

     6.0%                  12

     7.0%                  11

     8.0%                  10

     9.0%                   9

    10.0%                   8

<end output>
```

Task 3: **Permutation.**

Permutations in mathematics are the ways of picking several items from a large group, where order matters. For example, there are 12 ways to pick 2 items with different orders from a group of 4 items. In mathematics, the number of permutations of picking k items with different orders from a group of n items is denoted as $P_k^n$ , ${}_nP_k$, or $P(n,k)$.

$$P_k^n = \frac{n!}{(n-k)!}$$

Write a function named getNumberOfPermutations(n, k) for computing $P_k^n$ using following header:

**def** getNumberOfPermutations(n, k):

Write a test program that prompts the user to enter n and k and displays $P_k^n$ as shown in the following **sample run:**

<Output>

Enter n: 4

Enter k: 2

The P(n, k) is 12

<*End Output*>

<Output>

Enter n: 500

Enter k: 45

The P(n, k) is

369009161388133685496757481300458053847719140294230855904020031958668050285973462153715030171890388700653158400000000000

<*End Output*>

Optional/Bonus Task: **Office Snack Stock Tracker**
*This is a bonus question. Successfully completing this problem can earn you an additional 5 points. Not completing it will not affect your Project 1 grade.*

**Scenario:**

The office snack pantry is running low, and it's your job to track the snack stock levels. Employees can check how many snacks are available, take snacks, or restock them. You need to write a Python program to help manage the office snacks.

**Requirements:**

1. **Snack List (List)**:
   Create a list of available snacks with initial quantities (e.g., "Chips: 10", "Cookies: 5").
2. **Check Snack Stock (Loops and Lists)**:
   Allow the user (employee) to check how many of each snack is available.
3. **Take Snacks (Conditions and Lists)**:
   Allow the user to take snacks. Reduce the quantity of the chosen snack.
4. **Restock Snacks (Functions)**:
   Create a function to restock snacks by adding to the current quantity.
5. **Snack Usage Tracking (Mathematical Functions)**:
   Calculate how many snacks were taken and display the total number taken during the session.

**Example Output:**

Welcome to the Office Snack Stock Tracker!

Available snacks:

1. Chips: 10
2. Cookies: 5
3. Candy: 7

Would you like to take a snack or restock? (take/restock/check/exit): take

Which snack would you like? Chips

How many Chips would you like? 3

You have taken 3 Chips.

Available snacks:

1. Chips: 7
2. Cookies: 5
3. Candy: 7

Would you like to take a snack or restock? (take/restock/check/exit): exit

*Total snacks taken today: 3*

**Hints for Each Step:**

**Step 1: Snack List (List)**

*Hint*:

- You'll need a list or dictionary to store the snacks and their quantities.
- A dictionary might be a better choice, as it allows you to pair each snack with its available amount, like this:

E.g.: snacks = {"Chips": 10, "Cookies": 5, "Candy": 7}

**Step 2: Check Snack Stock (Loops and Lists)**

*Hint*:

- Use a for loop to go through the snack dictionary and print the snack name and quantity.
- You can loop through the dictionary like this:

e.g.: for snack, qty in snacks.items():

    print(f"{snack}: {qty}")

**Step 3: Take Snacks (Conditions and Lists)**

*Hint*:

- Ask the user which snack they want and how many they would like to take.
- First, check if the snack exists in your list (if snack in snacks:).
- Then, check if there are enough snacks available (if amount <= snacks[snack]:).
- Subtract the amount taken from the snack's quantity.

**Step 4: Restock Snacks (Functions)**

*Hint*:

- Define a function that takes the snack name and the amount to restock as inputs.
- Inside the function, add the restock amount to the current snack quantity:

e.g.: def restock(snack, amount):

snacks[snack] += amount

**Step 5: Snack Usage Tracking (Mathematical Functions)**

*Hint*:

- Create a variable total_taken and initialize it at 0.
- Each time a snack is taken, add the amount taken to this variable to keep track of how many snacks have been consumed in total.

e.g.: total_taken += amount

**Step 6: Main Loop (User Interaction)**

*Hint*:

- Use a while loop to keep the program running until the user types "exit."
- Inside the loop, ask the user what they want to do (e.g., take a snack, restock, check, or exit).
- Use if, elif, and else statements to handle the different actions based on the user's input.

**Step 7: Exit and Show Total Snacks Taken**

*Hint*:

- When the user types "exit", break out of the loop.
- Print the total number of snacks taken during the session by displaying the total_taken variable:

e.g.: print(f"Total snacks taken today: {total_taken}")

**Additional Tips:**

1. **User Input Validation**:
   If the user tries to take more snacks than available, let them know and don't allow the action.

2. **Case Sensitivity**:
   To make the program user-friendly, convert user input to lowercase or capitalize it where necessary so it's easier to match snack names.

3. **Error Handling**:
   Consider adding a check for invalid inputs (e.g., when a snack isn't in the list). You can use an else statement to handle this.

4. **Improve Readability**:
   Use comments throughout your code to explain what each section does. This makes the code easier to follow, especially for beginners.