**Best Programming Practices in C#**

1. Use meaningful variable and method names.
2. Follow PascalCase for class names and method names.
3. Follow camelCase for variable names.
4. Properly structure code with clear comments and modular methods.

**C# Code: Compute the Sum of Digits for a Random 4-Digit Number**

**Sample Program 1:** Create a program to find the sum of all the digits of a number given by a user using an array and display the sum.

1. Use Math.random() and get a 4-digit random integer number
2. Write a method to count digits in the number
3. Write a method to return an array of digits from a given number.
4. Write a method to Find the sum of the digits of the number in the array
5. Finally, display the sum of the digits of the number

using System;

class SumOfDigits

{

// Generate a 4-digit random number

public int Get4DigitRandomNumber()

{

Random random = new Random();

return random.Next(1000, 10000); // Generates a random number between 1000 and 9999

}

// Count the number of digits in a given number

public int CountDigits(int number)

{

int count = 0;

while (number > 0)

{

count++;

number /= 10;

}

return count;

}

// Extract digits from the number and store them in an array

public int[] GetDigits(int number, int count)

{

int[] digits = new int[count];

for (int i = count - 1; i >= 0; i--)

{

digits[i] = number % 10;

number /= 10;

}

return digits;

}

// Calculate the sum of the digits in the array

public int SumArray(int[] array)

{

int sum = 0;

foreach (int digit in array)

{

sum += digit;

}

return sum;

}

static void Main(string[] args)

{

// Instantiate the SumOfDigits class

SumOfDigits sumOfDigits = new SumOfDigits();

// Generate a 4-digit random number

int number = sumOfDigits.Get4DigitRandomNumber();

Console.WriteLine("The Random Number is: " + number);

// Count the number of digits

int count = sumOfDigits.CountDigits(number);

Console.WriteLine("The Count of Digits is: " + count);

// Extract digits into an array

int[] digits = sumOfDigits.GetDigits(number, count);

// Calculate the sum of the digits

int sum = sumOfDigits.SumArray(digits);

// Display the sum of the digits

Console.WriteLine("Sum of Digits: " + sum);

}

}



# Level 3 Practice Programs

1. Create a program to find the shortest, tallest, and mean height of players present in a football team.

**Hint =>**

a. The formula to calculate the mean is: mean = sum of all elements/number of elements

b. Create an int array named heights of size 11 and get 3 digits random height in cms for each player in the range 150 cms to 250 cms

c. Write the method to Find the sum of all the elements present in the array.

d. Write the method to find the mean height of the players on the football team

e. Write the method to find the shortest height of the players on the football team

f. Write the method to find the tallest height of the players on the football team

g. Finally display the results

| using System;  class Solution {  // Calculate the sum of heights  public static int CalculateSum(int[] heights) {  int sum = 0;  foreach (int height in heights) {  sum += height;  }  return sum;  }   // Calculate the mean height  public static double CalculateMean(int[] heights) {  if (heights.Length == 0) return 0;    int sum = CalculateSum(heights);    return (double)sum / heights.Length;  }   // Find the shortest height  public static int FindShortest(int[] heights) {  if (heights.Length == 0) return 0;    int shortest = Int32.MaxValue;    foreach (int height in heights) {  if (height < shortest) {  shortest = height;  }  }  return shortest;  }   // Find the tallest height  public static int FindTallest(int[] heights) {  if (heights.Length == 0) return 0;    int tallest = Int32.MinValue;    foreach (int height in heights) {  if (height > tallest) {  tallest = height;  }  }  return tallest;  }   public static void Main() {  // Constants for height range and number of players  const int MinHeight = 150;  const int MaxHeight = 250;  const int PlayerCount = 11;   // Generate random heights  Random random = new Random();  int[] heights = new int[PlayerCount];  for (int i = 0; i < PlayerCount; i++) {  heights[i] = random.Next(MinHeight, MaxHeight + 1);  }   // Display results  Console.WriteLine("Heights of players: " + string.Join(", ", heights));  Console.WriteLine("Shortest: {0} cm", FindShortest(heights));  Console.WriteLine("Tallest: {0} cm", FindTallest(heights));  Console.WriteLine("Mean: {0:F2} cm", CalculateMean(heights));  } } |
| --- |

2. Extend or Create a ***NumberChecker*** utility class and perform the following task. Call from the main() method the different methods and display results. Make sure all are static methods

**Hint =>**

a. Method to Find the count of digits in the number

b. Method to Store the digits of the number in a digits array

c. Method to Check if a number is a duck number using the digits array. A duck number is a number that has a non-zero digit present in it

d. Method to check if the number is an armstrong number using the digits array. ​​Armstrong number is a number that is equal to the sum of its own digits raised to the power of the number of digits. Eg: 153 = 1^3 + 5^3 + 3^3

e. Method to find the largest and second largest elements in the digits array. Use ***Int32.MinValue*** to initialize the variable.

f. Method to find the smallest and second smallest elements in the digits array. Use ***Int32.MaxValue*** to initialize the variable.

| using System;  class Solution {  // Extract digits of a number as an array  public static int[] GetDigits(int number) {  int length = 0, tempNumber = number;  while (tempNumber != 0) {  length++;  tempNumber /= 10;  }  int[] digits = new int[length];  int i = length - 1;  while (number != 0) {  digits[i] = number % 10;  i--;  number /= 10;  }  return digits;  }   // Check if a number is a Duck Number  public static bool IsDuckNumber(int number) {  int[] digits = GetDigits(number);  if (digits[0] == 0) return false;  foreach (int digit in digits) {  if (digit == 0) return true;  }  return false;  }   // Check if a number is an Armstrong Number  public static bool IsArmstrongNumber(int number) {  int[] digits = GetDigits(number);  int power = digits.Length;  int sum = 0;  foreach (int digit in digits) {  sum += (int)Math.Pow(digit, power);  }  return sum == number;  }   // Find the largest and second-largest digits  public static void FindLargestAndSecondLargest(int[] digits) {  int largest = int.MinValue, secondLargest = int.MinValue;  foreach (int digit in digits) {  if (digit > largest) {  secondLargest = largest;  largest = digit;  } else if (digit > secondLargest) {  secondLargest = digit;  }  }  Console.WriteLine("Largest: {0}, Second Largest: {1}", largest, secondLargest);  }   // Find the smallest and second-smallest digits  public static void FindSmallestAndSecondSmallest(int[] digits) {  int smallest = int.MaxValue, secondSmallest = int.MaxValue;  foreach (int digit in digits) {  if (digit < smallest) {  secondSmallest = smallest;  smallest = digit;  } else if (digit < secondSmallest) {  secondSmallest = digit;  }  }  Console.WriteLine("Smallest: {0}, Second Smallest: {1}", smallest, secondSmallest);  }   // Main Method  public static void Main() {  Console.Write("Enter a number: ");  int number = Convert.ToInt32(Console.ReadLine());   Console.WriteLine("Is Duck Number: {0}", IsDuckNumber(number));  Console.WriteLine("Is Armstrong Number: {0}", IsArmstrongNumber(number));   int[] digits = GetDigits(number);  FindLargestAndSecondLargest(digits);  FindSmallestAndSecondSmallest(digits);  } } |
| --- |

3. Extend or Create a ***NumberChecker*** utility class and perform the following task. Call from the main() method the different methods and display results. Make sure all are static methods

**Hint =>**

a. Method to find the count of digits in the number and a Method to Store the digits of the number in a digits array

b. Method to find the sum of the digits of a number using the digits array

c. Method to find the sum of the squares of the digits of a number using the digits array. Use ***Math.Pow()*** method

d. Method to Check if a number is a Harshad number using a digits array. A number is called a Harshad number if it is divisible by the sum of its digits. For e.g. 21

e. Method to find the frequency of each digit in the number. Create a 2D array to store the frequency with digit in the first column and frequency in the second column.

| using System;  class Solution {  // Method to get digits of a number as an array  public static int[] GetDigits(int number) {  int length = 0, tempNumber = number;  while (tempNumber != 0) {  length++;  tempNumber /= 10;  }   int[] digits = new int[length];  int i = 0;  while (number != 0) {  digits[i] = number % 10;  i++;  number /= 10;  }   return digits;  }   // Method to calculate the sum of digits of a number  public static int GetSum(int number) {  int[] digits = GetDigits(number);  int sum = 0;  foreach (int digit in digits) {  sum += digit;  }  return sum;  }   // Method to calculate the sum of squares of digits of a number  public static double GetSumOfSquaresOfDigits(int number) {  int[] digits = GetDigits(number);  double sumSquares = 0;  foreach (int digit in digits) {  sumSquares += Math.Pow(digit, 2);  }  return sumSquares;  }   // Method to check if a number is a Harshad number  public static bool CheckHarshadNumber(int number) {  int sum = GetSum(number);  return (number % sum == 0);  }   // Method to get frequency of digits  public static int[,] GetDigitFrequency(int number) {  int[] digitsArray = GetDigits(number);  int[] frequencyArray = new int[10];   // Count the frequency of each digit  foreach (int digit in digitsArray) {  frequencyArray[digit]++;  }   // Determine the number of unique digits  int uniqueDigitCount = 0;  for (int i = 0; i < frequencyArray.Length; i++) {  if (frequencyArray[i] > 0) {  uniqueDigitCount++;  }  }   // Create a 2D array for digit and frequency  int[,] result = new int[uniqueDigitCount, 2];  int index = 0;   // Populate the result array  for (int i = 0; i < frequencyArray.Length; i++) {  if (frequencyArray[i] > 0) {  result[index, 0] = i;  result[index, 1] = frequencyArray[i];  index++;  }  }  return result;  }    // Main method to test the utility  public static void Main() {  Console.Write("Enter a number: ");  int number = Convert.ToInt32(Console.ReadLine());   // Calculate and display results  Console.WriteLine("Sum of digits: {0}", GetSum(number));  Console.WriteLine("Sum of squares of digits: {0}", GetSumOfSquaresOfDigits(number));  Console.WriteLine("Is Harshad Number: {0}", CheckHarshadNumber(number));   // Get and display digit frequency  int[,] frequencyArray = GetDigitFrequency(number);  Console.WriteLine("Digit Frequencies:");  for (int i = 0; i < frequencyArray.GetLength(0); i++) {  Console.WriteLine("Digit: {0}, Frequency: {1}", frequencyArray[i, 0], frequencyArray[i, 1]);  }  } } |
| --- |

4. Extend or Create a ***NumberChecker*** utility class and perform the following task. Call from the main() method the different methods and display results. Make sure all are static methods

**Hint =>**

a. Method to find the count of digits in the number and a Method to Store the digits of the number in a digits array

b. Method to reverse the digits array

c. Method to compare two arrays and check if they are equal

d. Method to check if a number is a palindrome using the Digits. A palindrome number is a number that remains the same when its digits are reversed.

e. Method to Check if a number is a duck number using the digits array. A duck number is a number that has a non-zero digit present in it

| using System;  class Solution {  // Method to count the number of digits in a number  public static int GetDigitCount(int number) {  int length=0;  while (number != 0) {  length++;  number /= 10;  }  return length;  }   // Method to get the digits of a number as an array  public static int[] GetDigits(int number) {  int length = GetDigitCount(number);  int[] digits = new int[length];  int i = length - 1;   while (number != 0) {  digits[i] = number % 10;  i--;  number /= 10;  }   return digits;  }   // Method to reverse the digits array  public static int[] ReverseDigitsArray(int[] digitsArray) {  return digitsArray.Reverse().ToArray();  }   // Method to check if two arrays are equal  public static bool CheckArraysEqual(int[] array1, int[] array2) {  return array1.SequenceEqual(array2);  }   // Method to check if a number is a palindrome  public static bool CheckPalindrome(int number) {  int[] digitsArray = GetDigits(number);  int[] reversedArray = ReverseDigitsArray(digitsArray);    return CheckArraysEqual(digitsArray, reversedArray);  }   // Method to check if a number is a duck number  public static bool CheckDuckNumber(int number) {  int[] digits = GetDigits(number);  foreach (int digit in digits) {  if (digit == 0) return true;  }  return false;  }   // Main method  public static void Main() {  Console.Write("Enter a number: ");  int number = Convert.ToInt32(Console.ReadLine());   int[] digitsArray = GetDigits(number);  int[] digitsArrayReversed = ReverseDigitsArray(digitsArray);   Console.WriteLine("Digits array is: {0}", string.Join(",", digitsArray));  Console.WriteLine("Reversed digits array is: {0}", string.Join(",", digitsArrayReversed));  Console.WriteLine("Is the digits array equal to its reverse? {0}", CheckArraysEqual(digitsArray, digitsArrayReversed));  Console.WriteLine("Is the number a palindrome? {0}", CheckPalindrome(number));  Console.WriteLine("Is the number a Duck Number? {0}", CheckDuckNumber(number));  } } |
| --- |

5. Extend or Create a ***NumberChecker*** utility class and perform the following task. Call from the main() method the different methods and display results. Make sure all are static methods

**Hint =>**

a. Method to Check if a number is a prime number. A prime number is a number greater than 1 that has no positive divisors other than 1 and itself.

b. Method to Check if a number is a neon number. A neon number is a number where the sum of digits of the square of the number is equal to the number itself

c. Method to Check if a number is a spy number. A number is called a spy number if the sum of its digits is equal to the product of its digits

d. Method to Check if a number is an automorphic number. An automorphic number is a number whose square ends with the number itself. E.g. 5 is an automorphic number

e. Method to Check if a number is a buzz number. A buzz number is a number that is either divisible by 7 or ends with 7

| using System;  class Solution {    // Method to get the digits of a number as an array  public static int[] GetDigits(int number) {  int length = 0, tempNumber = number;  while (tempNumber != 0) {  length++;  tempNumber /= 10;  }  int[] digits = new int[length];  int i = 0;  while (number != 0) {  int digit = number % 10;  digits[length - 1 - i] = digit;  i++;  number /= 10;  }  return digits;  }   // Method to get the sum of the digits of a number  public static int GetSum(int number) {  int[] digits = GetDigits(number);  int sum = 0;  foreach (int digit in digits) {  sum += digit;  }  return sum;  }   // Method to get the product of the digits of a number  public static int GetProduct(int number) {  int[] digits = GetDigits(number);  int product = 1;  foreach (int digit in digits) {  product \*= digit;  }  return product;  }   // Method to check if a number is a Neon number  public static bool CheckNeonNumber(int number) {  int squareNumber = (int)Math.Pow(number, 2);  int squareSum = GetSum(squareNumber);    return number == squareSum;  }   // Method to check if a number is a Spy number  public static bool CheckSpyNumber(int number) {  int product = GetProduct(number);  int sum = GetSum(number);    return sum == product;  }   // Method to check if a number is an Automorphic number  public static bool CheckAutomorphicNumber(int number) {  int squareNumber = (int)Math.Pow(number, 2);  while (number > 0) {  int numDigit = number % 10;  int squareDigit = squareNumber % 10;    if (numDigit != squareDigit) return false;    number /= 10;  squareNumber /= 10;  }  return true;  }   // Method to check if a number is a Buzz number  public static bool CheckBuzzNumber(int number) {  return number % 7 == 0 || number % 10 == 7;  }   // Method to check if a number is Prime  public static bool CheckPrime(int number) {  if (number < 2) return false;  for (int i = 2; i <= Math.Sqrt(number); i++) {  if (number % i == 0) return false;  }  return true;  }   // Main method  public static void Main() {  Console.Write("Enter a number: ");  int number = Convert.ToInt32(Console.ReadLine());   int[] digitsArray = GetDigits(number);   Console.WriteLine("Digits array is: {0}", string.Join(",", digitsArray));  Console.WriteLine("Is the number prime? {0}", CheckPrime(number));  Console.WriteLine("Is the number a Neon number? {0}", CheckNeonNumber(number));  Console.WriteLine("Is the number a Spy number? {0}", CheckSpyNumber(number));  Console.WriteLine("Is the number an Automorphic number? {0}", CheckAutomorphicNumber(number));  Console.WriteLine("Is the number a Buzz number? {0}", CheckBuzzNumber(number));  } } |
| --- |

6. Extend or Create a ***NumberChecker*** utility class and perform the following task. Call from the main() method the different methods and display results. Make sure all are static methods

**Hint =>**

a. Method to find factors of a number and return them as an array. Note there are 2 for loops one for the count and another for finding the factor and storing in the array

b. Method to find the greatest factor of a Number using the factors array

c. Method to find the sum of the factors using factors array and return the sum

d. Method to find the product of the factors using factors array and return the product

e. Method to find product of cube of the factors using the factors array. Use ***Math.Pow()***

f. Method to Check if a number is a perfect number. Perfect numbers are positive integers that are equal to the sum of their proper divisors

g. Method to find the number is an abundant number. A number is called an abundant number if the sum of its proper divisors is greater than the number itself

h. Method to find the number is a deficient number. A number is called a deficient number if the sum of its proper divisors is less than the number itself

i. Method to Check if a number is a strong number. A number is called a strong number if the sum of the factorial of its digits is equal to the number itself

| using System;  class Solution {  // Method to find factors of a number and return them as an array  public static int[] FindFactors(int number) {  int count = 0;   // Count the number of factors  for (int i = 1; i <= number; i++) {  if (number % i == 0) count++;  }   int[] factors = new int[count];  int index = 0;   // Store the factors in an array  for (int i = 1; i <= number; i++) {  if (number % i == 0) {  factors[index] = i;  }  index++;  }   return factors;  }   // Method to find the greatest factor of a number  public static int FindGreatestFactor(int number) {  int[] factors = FindFactors(number);    return factors[factors.Length - 1];  }   // Method to find the sum of the factors  public static int SumOfFactors(int number) {  int[] factors = FindFactors(number);  int sum = 0;   foreach (int factor in factors) {  sum += factor;  }   return sum;  }   // Method to find the product of the factors  public static long ProductOfFactors(int number) {  int[] factors = FindFactors(number);  long product = 1;   foreach (int factor in factors) {  product \*= factor;  }   return product;  }     // Method to find the product of the cubes of the factors  public static double ProductOfCubesOfFactors(int number) {  int[] factors = FindFactors(number);  double product = 1;   foreach (int factor in factors) {  product \*= Math.Pow(factor, 3);  }   return product;  }   // Helper Method to calculate proper divisor sum of number  public static bool ProperDivisorSum(int number) {  int[] factors = FindFactors(number);  int sum = 0;    for (int i = 0; i < factors.Length - 1; i++) {  sum += factors[i];  }   return sum;  }    // Method to check if a number is a perfect number  public static bool IsPerfectNumber(int number) {  int sum = ProperDivisorSum(number);   return sum == number;  }   // Method to check if a number is an abundant number  public static bool IsAbundantNumber(int number) {  int sum = ProperDivisorSum(number);   return sum > number;  }   // Method to check if a number is a deficient number  public static bool IsDeficientNumber(int number) {  int sum = ProperDivisorSum(number);   return sum < number;  }   // Helper method to calculate the factorial of a number  private static int Factorial(int n) {  int result = 1;  for (int i = 2; i <= n; i++) {  result \*= i;  }  return result;  }   // Method to check if a number is a strong number  public static bool IsStrongNumber(int number) {  int temp = number;  int sum = 0;   while (temp > 0) {  int digit = temp % 10;  sum += Factorial(digit);  temp /= 10;  }   return sum == number;  }     // Main method to call the utility methods and display results  public static void Main() {  Console.Write("Enter a number: ");  int number = Convert.ToInt32(Console.ReadLine());   int[] factors = FindFactors(number);  Console.WriteLine("Factors: " + string.Join(", ", factors));  Console.WriteLine("Greatest Factor: " + FindGreatestFactor(number));  Console.WriteLine("Sum of Factors: " + SumOfFactors(number));  Console.WriteLine("Product of Factors: " + ProductOfFactors(number));  Console.WriteLine("Product of Cubes of Factors: " + ProductOfCubesOfFactors(number));  Console.WriteLine("Is Perfect Number: " + IsPerfectNumber(number));  Console.WriteLine("Is Abundant Number: " + IsAbundantNumber(number));  Console.WriteLine("Is Deficient Number: " + IsDeficientNumber(number));  Console.WriteLine("Is Strong Number: " + IsStrongNumber(number));  } } |
| --- |

7. Write a program to generate a six-digit OTP number using Math.Random() method. Validate the numbers are unique by generating the OTP number 10 times and ensuring all the 10 OTPs are not the same

**Hint =>**

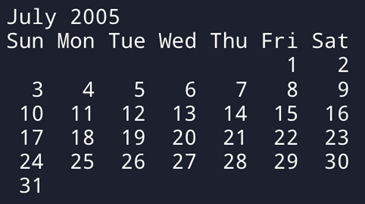
a. Write a method to Generate a 6-digit OTP number using Math.Random()

b. Create an array to save the OTP numbers generated 10 times

c. Write a method to ensure that the OTP numbers generated are unique. If unique return true else return false

| using System;  class Solution {  // Method to generate a 6-digit OTP  public static int GenerateOTP() {  Random random = new Random();  return random.Next(100000, 1000000);  }   // Method to check if all OTPs in the array are unique  public static bool CheckOTPsUnique(int[] otps) {  for (int i = 0; i < otps.Length; i++) {  for (int j = i + 1; j < otps.Length; j++) {  if (otps[i] == otps[j]) {  return false;  }  }  }  return true;  }   public static void Main() {  int[] otps = new int[10];   // Generate 10 OTPs  for (int i = 0; i < 10; i++) {  otps[i] = GenerateOTP();  }   // Display the generated OTPs  Console.WriteLine("Generated OTPs: {0}", string.Join(", ", otps));   // Check if all OTPs are unique  bool unique = CheckOTPsUnique(otps);   Console.WriteLine("Are all OTPs unique? {0}" + unique);  } } |
| --- |

8. Create a program to display a calendar for a given month and year. The program should take the month and year as input from the user and display the calendar for that month. E.g. for 07 2005 user input, the program should display the calendar as shown below



**Hint =>**

a. Write a Method to get the name of the month. For this define a month Array to store the names of the months

b. Write a Method to get the number of days in the month. For this define a days Array to store the number of days in each month. For Feb month, check for Leap Year to get the number of days. Also, define a Leap Year Method.

c. Write a method to get the first day of the month using the Gregorian calendar algorithm

y0 = y − (14 − m) / 12

x = y0 + y0/4 − y0/100 + y0/400

m0 = m + 12 × ((14 − m) / 12) − 2

d0 = (d + x + 31m0 / 12) mod 7

d. Displaying the Calendar requires 2 ***for*** loops.

e. The first ***for*** loop up to the first day to get the proper indentation. As in the example above 3 spaces from Sun to Thu as to be set as July 1st starts on Fri

f. The Second ***for*** loop Displays the days of the month starting from 1 to the number of days. Add proper indentation for single-digit days using ***%3d*** to display the integer right-justified in a field of width 3. Please note to move to the next line after Sat

| using System;  class Solution {  public static string GetMonthName(int month) {  string[] monthNames = { "January", "February", "March", "April", "May", "June",  "July", "August", "September", "October", "November", "December" };    return monthNames[month - 1];  }    public static bool CheckLeapYear(int year) {  return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);  }    public static int GetDaysInMonth(int month, int year) {  if (month == 2) {  return CheckLeapYear(year) ? 29 : 28;  }  int[] days = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };    return days[month - 1];  }    public static int GetFirstDayOfMonth(int month, int year) {  int y0 = year - (14 - month) / 12;  int x = y0 + y0 / 4 - y0 / 100 + y0 / 400;  int m0 = month + 12 \* ((14 - month) / 12) - 2;    return (1 + x + (31 \* m0) / 12) % 7;  }   public static void DisplayCalendar(int month, int year) {  Console.WriteLine("{0} {1}" , GetMonthName(month) , year);  Console.WriteLine("Sun Mon Tue Wed Thu Fri Sat");   int days = GetDaysInMonth(month, year);  int firstDay = GetFirstDayOfMonth(month, year);   for (int i = 0; i < firstDay; i++) {  Console.Write(" ");  }   for (int day = 1; day <= days; day++) {  Console.Write("{0,3} ", day);  if ((day + firstDay) % 7 == 0) Console.WriteLine();  }  Console.WriteLine();  }   static void Main(string[] args) {  Console.WriteLine("Enter month (1-12): ");  int month = Convert.ToInt32(Console.ReadLine());  Console.WriteLine("Enter year: ");  int year = Convert.ToInt32(Console.ReadLine());   DisplayCalendar(month, year);  } } |
| --- |

9. Write a program Euclidean distance between two points as well as the equation of the line using those two points. Use Math functions ***Math.Pow()*** and ***Math.Sqrt()***

**Hint =>**

a. Take inputs for 2 points x1, y1, and x2, y2

b. Method to find the Euclidean distance between two points and return the distance

distance = (x2-x1)2 +(y2-y1)2

d. Write a Method to find the equation of a line given two points and return the equation which includes the slope and the y-intercept

The equation of a line is given by the equation y = m\*x + b Where m is the slope and b is the y-intercept. So firstly compute the slope using the formulae

m = (y2 - y1)/(x2 - x1)

Post that compute the y-intercept b using the formulae

b = y1 - m\*x1

Finally, return an array having slope m and y-intercept b

| using System;  class Solution {  // Method to calculate the Euclidean distance between two points  public static double CalculateDistance(double x1, double y1, double x2, double y2) {  double distance = Math.Sqrt(Math.Pow(x2 - x1, 2) + Math.Pow(y2 - y1, 2));    return distance;  }   // Method to calculate the slope and y-intercept of a line passing through two points  public static double[] GetLineEquation(double x1, double y1, double x2, double y2) {  double slope = (y2 - y1) / (x2 - x1);  double yIntercept = y1 - slope \* x1;    return new double[] { slope, yIntercept };  }   public static void Main() {  // Input for two points  Console.Write("Enter x1: ");  double x1 = Convert.ToDouble(Console.ReadLine());  Console.Write("Enter y1: ");  double y1 = Convert.ToDouble(Console.ReadLine());  Console.Write("Enter x2: ");  double x2 = Convert.ToDouble(Console.ReadLine());  Console.Write("Enter y2: ");  double y2 = Convert.ToDouble(Console.ReadLine());   // Calculate distance  double distance = CalculateDistance(x1, y1, x2, y2);  Console.WriteLine("Euclidean Distance: {0}" , distance);   // Calculate line equation  double[] lineEquation = GetLineEquation(x1, y1, x2, y2);  Console.WriteLine("Equation of the Line: y = {0} \* x + {1}" , lineEquation[0] ,lineEquation[1]);  } } |
| --- |

10. Write a program to find the 3 points that are collinear using the slope formulae and area of triangle formulae. check A (2, 4), B (4, 6) and C (6, 8) are Collinear for sampling.

**Hint =>**

a. Take inputs for 3 points x1, y1, x2, y2, and x3, y3

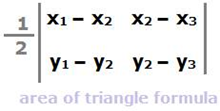
b. Write a Method to find the 3 points that are collinear using the slope formula. The 3 points A(x1,y1), b(x2,y2), and c(x3,y3) are collinear if the slopes formed by 3 points ab, bc, and cd are equal.

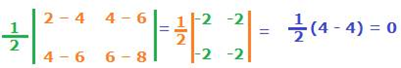
slope AB = (y2 - y1)/(x2 - x1), slope BC = (y3 - y2)/(x3 - x3)

slope AC = (y3 - y1)/(x3 - x1) Points are collinear if

slope AB = slope BC = slope Ac

c. The method to find the three points is collinear using the area of the triangle formula. The Three points are collinear if the area of the triangle formed by three points is 0. The area of a triangle is





area = 0.5 \* (x1 \* (y2 - y3) + x2 \* (y3 - y1) + x3 \* (y1 - y2))

| using System;  class Solution {  // Method to check collinearity using the slope formula  public static bool CheckCollinearUsingSlope(double x1, double y1, double x2, double y2, double x3, double y3) {  double slopeAB = (y2 - y1) / (x2 - x1);  double slopeBC = (y3 - y2) / (x3 - x2);  double slopeAC = (y3 - y1) / (x3 - x1);   return (slopeAB == slopeAC && slopeAB == slopeBC);  }   // Method to check collinearity using the area of the triangle formula  public static bool CheckCollinearUsingArea(double x1, double y1, double x2, double y2, double x3, double y3) {  double area = 0.5 \* (x1 \* (y2 - y3) + x2 \* (y3 - y1) + x3 \* (y1 - y2));  return Math.Abs(area) == 0;  }   public static void Main() {  // Input for three points  Console.Write("Enter x1, y1: ");  double x1 = Convert.ToDouble(Console.ReadLine());  double y1 = Convert.ToDouble(Console.ReadLine());    Console.Write("Enter x2, y2: ");  double x2 = Convert.ToDouble(Console.ReadLine());  double y2 = Convert.ToDouble(Console.ReadLine());    Console.Write("Enter x3, y3: ");  double x3 = Convert.ToDouble(Console.ReadLine());  double y3 = Convert.ToDouble(Console.ReadLine());   // Check collinearity using slope formula  bool collinearSlope = CheckCollinearUsingSlope(x1, y1, x2, y2, x3, y3);  Console.WriteLine("Collinear using Slope Formula: {0}" , collinearSlope);   // Check collinearity using area formula  bool collinearArea = CheckCollinearUsingArea(x1, y1, x2, y2, x3, y3);  Console.WriteLine("Collinear using Area Formula: {0}" , collinearArea);  } } |
| --- |

11. Create a program to find the bonus of 10 employees based on their years of service as well as the total bonus amount the 10-year-old company Zara has to pay as a bonus, along with the old and new salary.

**Hint =>**

a. Zara decides to give a bonus of 5% to employees whose year of service is more than 5 years or 2% if less than 5 years

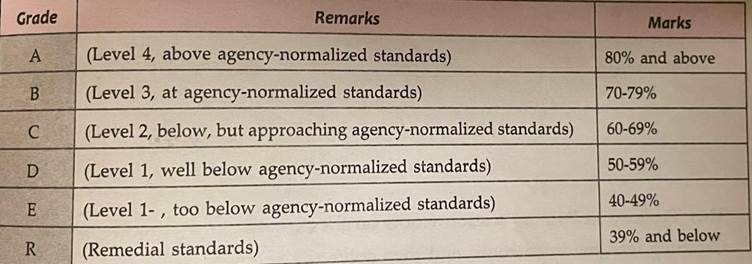
b. Create a Method to determine the Salary and years of service and return the same. Use the ***Math.Random()*** method to determine the 5-digit salary for each employee and also use the random method to determine the years of service. Define 2D Array to save the salary and years of service.

c. Write a Method to calculate the new salary and bonus based on the logic defined above and return the new 2D Array of the latest salary and bonus amount

d. Write a Method to Calculate the sum of the Old Salary, the Sum of the New Salary, and the Total Bonus Amount and display it in a Tabular Format

| using System;  public class Solution {  // Generates random salaries (5-digit) and years of service for employees  public static double[,] GenerateEmployeeData(int numEmployees) {  double[,] employeeData = new double[numEmployees, 2];  Random random = new Random();   for (int i = 0; i < numEmployees; i++) {  // Generate a random 5-digit salary  employeeData[i, 0] = random.Next(10000, 100000);   // Generate random years of service  employeeData[i, 1] = random.Next(1, 11);  }   return employeeData;  }   // Calculates the bonus and new salary for each employee  public static double[,] CalculateBonusAndNewSalary(double[,] employeeData, int numEmployees) {  double[,] updatedData = new double[numEmployees, 2];   for (int i = 0; i < numEmployees; i++) {  // Extract salary and years of service    double salary = employeeData[i, 0];  double yearsOfService = employeeData[i, 1];  double bonus;   // Calculate bonus based on years of service  if (yearsOfService > 5) {  bonus = salary \* 0.05;  }  else {  bonus = salary \* 0.02;  }   // Calculate new salary  double newSalary = salary + bonus;   // Store bonus and new salary in the array  updatedData[i, 0] = bonus;  updatedData[i, 1] = newSalary;  }   return updatedData;  }   // Displays the employee details along with total salary and bonus amounts  public static void DisplayResults(double[,] employeeData, double[,] updatedData, int numEmployees) {  double totalOldSalary = 0, totalBonus = 0, totalNewSalary = 0;   // Print table header  Console.WriteLine("Employee\tSalary\t\tYears\t\tBonus\t\tNew Salary");   for (int i = 0; i < numEmployees; i++){  // Extract details  double salary = employeeData[i, 0];  double yearsOfService = employeeData[i, 1];  double bonus = updatedData[i, 0];  double newSalary = updatedData[i, 1];   // Print details in a single line  Console.WriteLine("{0}\t\t{1:F2}\t{2}\t\t{3:F2}\t\t{4:F2}", i + 1, salary, yearsOfService, bonus, newSalary);   // Update totals  totalOldSalary += salary;  totalBonus += bonus;  totalNewSalary += newSalary;  }   // Print totals  Console.WriteLine("Total\t\t{0:F2}\t\t\t{1:F2}\t{2:F2}", totalOldSalary, totalBonus, totalNewSalary); }     public static void Main() {  // Number of employees  int numEmployees = 10;   // Generate random salary and years of service for employees  double[,] employeeData = GenerateEmployeeData(numEmployees);   // Calculate bonuses and new salaries  double[,] updatedData = CalculateBonusAndNewSalary(employeeData, numEmployees);   // Display results and totals  DisplayResults(employeeData, updatedData, numEmployees);  } } |
| --- |

12. Create a program to take input marks of students in 3 subjects physics, chemistry, and maths. Compute the total, average, and the percentage score



**Hint =>**

a. Take input for the number of students

b. Write a method to generate random 2-digit scores for Physics, Chemistry, and Math (PCM) for the students and return the scores. This method returns a 2D array with PCM scores for all students

c. Write a Method to calculate the total, average, and percentages for each student and return a 2D array with the corresponding values. Please ensure to round off the values to 2 Digits using the ***Math.Round()*** method.

d. Finally, write a Method to display the scorecard of all students with their scores, total, average, and percentage in a tabular format using ***"\t"***.

| using System;  class Solution {  // Method to generate random marks for Physics, Chemistry, and Mathematics for each student  public static int[,] GenerateMarks(int numStudents) {  int[,] marks = new int[numStudents, 3];  Random random = new Random();   for (int i = 0; i < numStudents; i++) {  // Generate random marks for Physics, Chemistry, and Mathematics  marks[i, 0] = random.Next(50, 101);   marks[i, 1] = random.Next(50, 101);   marks[i, 2] = random.Next(50, 101);   }  return marks;  }   // Method to calculate the total, average, and percentage for each student  public static double[,] CalculateResults(int[,] marks, int numStudents) {  double[,] results = new double[numStudents, 3];    for (int i = 0; i < numStudents; i++) {  int totalMarks = marks[i, 0] + marks[i, 1] + marks[i, 2];  double averageMarks = totalMarks / 3.0;   double percentage = Math.Round((totalMarks / 300.0) \* 100, 2);   results[i, 0] = totalMarks;  results[i, 1] = Math.Round(averageMarks, 2);   results[i, 2] = percentage;  }   return results;  }   // Method to display the scorecard in a tabular format  public static void DisplayResults(int[,] marks, double[,] results, int numStudents) {  // Display table headers  Console.WriteLine("Student\tPhysics\tChemistry\tMathematics\tTotal\t\tAverage\t\tPercentage");   for (int i = 0; i < numStudents; i++) {  // Extract individual student details  int physics = marks[i, 0];  int chemistry = marks[i, 1];  int mathematics = marks[i, 2];  double total = results[i, 0];  double average = results[i, 1];  double percentage = results[i, 2];   // Display the student results in a tabular format  Console.WriteLine($" {i + 1} \t{physics}\t{chemistry}\t\t{mathematics}\t\t{total}\t\t{average}\t\t{percentage}%");  }  }    public static void Main() {  // Prompt for the number of students  Console.WriteLine("Enter the number of students: ");  int numStudents = Convert.ToInt32(Console.ReadLine());   // Generate random marks for Physics, Chemistry, and Mathematics  int[,] studentMarks = GenerateMarks(numStudents);   // Calculate total, average, and percentage for each student  double[,] studentResults = CalculateResults(studentMarks, numStudents);   // Display the scorecard in tabular format  DisplayResults(studentMarks, studentResults, numStudents);  } } |
| --- |

13. Write a program to perform matrix manipulation operations like addition, subtraction, multiplication, and transpose. Also finding the determinant and inverse of a matrix. The program should take random matrices as input and display the result of the operations.

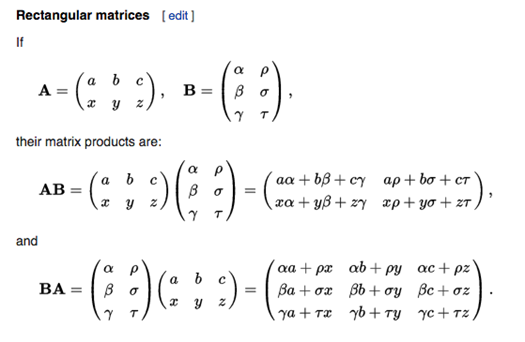
**Hint =>**

a. Write a Method to create a random matrix taking rows and columns as parameters

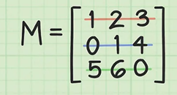
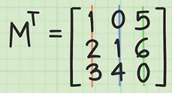
b. Write a Method to add two matrices

c. Write a Method to subtract two matrices

d. Write a Method to multiply two matrices

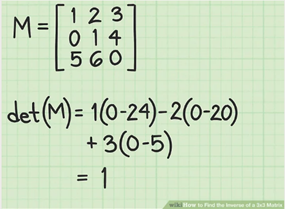
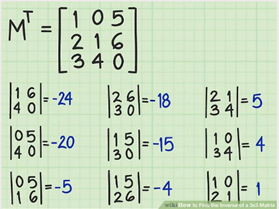


1. Write a Method to find the transpose of a matrix

e. Write a Method to find the determinant of a 2x2 matrix

f. Write a Method to find the determinant of a 3x3 matrix

g. Write a Method to find the inverse of a 2x2 matrix

h. Write a Method to find the inverse of a 3x3 matrix

i. Write a Method to display a matrix

| **using System;  class Solution {  // Create a random matrix with specified rows and columns  public static double[,] CreateRandomMatrix(int rows, int cols) {  Random rand = new Random();  double[,] matrix = new double[rows, cols];  for (int i = 0; i < rows; i++) {  for (int j = 0; j < cols; j++) {  matrix[i, j] = rand.Next(1, 10);  }  }  return matrix;  }   // Add two matrices  public static double[,] AddMatrices(double[,] A, double[,] B, int rows, int cols) {  double[,] result = new double[rows, cols];  for (int i = 0; i < rows; i++) {  for (int j = 0; j < cols; j++) {  result[i, j] = A[i, j] + B[i, j];   }  }  return result;  }   // Subtract two matrices  public static double[,] SubtractMatrices(double[,] A, double[,] B, int rows, int cols) {  double[,] result = new double[rows, cols];  for (int i = 0; i < rows; i++) {  for (int j = 0; j < cols; j++) {  result[i, j] = A[i, j] - B[i, j];  }  }  return result;  }   // Multiply two matrices  public static double[,] MultiplyMatrices(double[,] A, double[,] B, int rows, int colsA, int colsB) {  double[,] result = new double[rows, colsB];  for (int i = 0; i < rows; i++)  for (int j = 0; j < colsB; j++)  for (int k = 0; k < colsA; k++)  result[i, j] += A[i, k] \* B[k, j];  return result;  }   // Transpose a matrix  public static double[,] TransposeMatrix(double[,] matrix, int rows, int cols) {  double[,] result = new double[cols, rows];  for (int i = 0; i < rows; i++) {  for (int j = 0; j < cols; j++) {  result[j, i] = matrix[i, j];  }  }  return result;  }   // Determinant of a 2x2 matrix  public static double Determinant2x2(double[,] matrix) {  return matrix[0, 0] \* matrix[1, 1] - matrix[0, 1] \* matrix[1, 0];  }   // Determinant of a 3x3 matrix  public static double Determinant3x3(double[,] matrix) {  double a = matrix[0, 0], b = matrix[0, 1], c = matrix[0, 2];  double d = matrix[1, 0], e = matrix[1, 1], f = matrix[1, 2];  double g = matrix[2, 0], h = matrix[2, 1], i = matrix[2, 2];   return a \* (e \* i - f \* h) - b \* (d \* i - f \* g) + c \* (d \* h - e \* g);  }   // Inverse of a 2x2 matrix  public static double[,] Inverse2x2(double[,] matrix) {  double det = Determinant2x2(matrix);  if (det == 0) return null;  double[,] result = new double[2, 2];  result[0, 0] = matrix[1, 1] / det;  result[0, 1] = -matrix[0, 1] / det;  result[1, 0] = -matrix[1, 0] / det;  result[1, 1] = matrix[0, 0] / det;  return result;  }   // Inverse of a 3x3 matrix  public static double[,] Inverse3x3(double[,] matrix) {  double det = Determinant3x3(matrix);  if (det == 0) return null;  double[,] result = new double[3, 3];   result[0, 0] = (matrix[1, 1] \* matrix[2, 2] - matrix[1, 2] \* matrix[2, 1]) / det;  result[0, 1] = (matrix[0, 2] \* matrix[2, 1] - matrix[0, 1] \* matrix[2, 2]) / det;  result[0, 2] = (matrix[0, 1] \* matrix[1, 2] - matrix[0, 2] \* matrix[1, 1]) / det;   result[1, 0] = (matrix[1, 2] \* matrix[2, 0] - matrix[1, 0] \* matrix[2, 2]) / det;  result[1, 1] = (matrix[0, 0] \* matrix[2, 2] - matrix[0, 2] \* matrix[2, 0]) / det;  result[1, 2] = (matrix[0, 2] \* matrix[1, 0] - matrix[0, 0] \* matrix[1, 2]) / det;   result[2, 0] = (matrix[1, 0] \* matrix[2, 1] - matrix[1, 1] \* matrix[2, 0]) / det;  result[2, 1] = (matrix[0, 1] \* matrix[2, 0] - matrix[0, 0] \* matrix[2, 1]) / det;  result[2, 2] = (matrix[0, 0] \* matrix[1, 1] - matrix[0, 1] \* matrix[1, 0]) / det;   return result;  }   // Display matrix  public static void DisplayMatrix(double[,] matrix, int rows, int cols, string label = "Matrix") {  Console.WriteLine(label);  for (int i = 0; i < rows; i++)  {  for (int j = 0; j < cols; j++)  Console.Write(matrix[i, j] + "\t");  Console.WriteLine();  }  Console.WriteLine();  }   public static void Main() {  // Define matrix dimensions  int rows = 3;  int cols = 3;   // Create random 3x3 matrices  double[,] matrixA = CreateRandomMatrix(rows, cols);  double[,] matrixB = CreateRandomMatrix(rows, cols);   // Display matrices  Console.WriteLine("Matrix A:");  DisplayMatrix(matrixA, rows, cols);   Console.WriteLine("Matrix B:");  DisplayMatrix(matrixB, rows, cols);   // Perform operations and display results  DisplayMatrix(AddMatrices(matrixA, matrixB, rows, cols), rows, cols, "A + B");  DisplayMatrix(SubtractMatrices(matrixA, matrixB, rows, cols), rows, cols, "A - B");  DisplayMatrix(MultiplyMatrices(matrixA, matrixB, rows, cols, cols), rows, cols, "A \* B");  DisplayMatrix(TransposeMatrix(matrixA, rows, cols), rows, cols, "Transpose of A");   // For 3x3 matrices, calculate determinant and inverse  Console.WriteLine("Determinant of A: " + Determinant3x3(matrixA));  DisplayMatrix(Inverse3x3(matrixA), 3, 3, "Inverse of A");  } }** |
| --- |