Part1

Problem1

```
using System;
class Program
{
   static void Main()
   {
      int[] a = new int[3];
      a[0] = 10;
      a[1] = 20;
      a[2] = 30;

      int[] b = new int[] { 40, 50, 60 };

      int[] c = { 70, 80, 90 };

      Console.WriteLine(a[0]);
      Console.WriteLine(a[1]);
      Console.WriteLine(a[2]);

      Console.WriteLine(b[0]);
      Console.WriteLine(b[1]);
      Console.WriteLine(b[2]);

      Console.WriteLine(c[0]);
      Console.WriteLine(c[1]);
```

```
    Console.WriteLine(c[2]);


    Console.WriteLine("هنجرب خطأ:");
    try
    {
      Console.WriteLine(a[3]); // ده هيرمي خطأ لأن مفيش رقم 3 index
    }
    catch
    {
      Console.WriteLine("غلط! بتحاول توصل لمكان مش موجود في المصفوفة.");
    }
  }
}
```

Question1

من نوع بيانات معين بدون ما تديها قيم، كل عنصر فيها بياخد **القيمة الافتراضية** (Array) ، لما تنشئ مصفوفة#C في حسب نوع البيانات **(Default Value)**.

---

Problem2:

```
using System;


class Program
{
  static void Main()
  {
    int[] arr1 = { 1, 2, 3 };
```

```
        int[] arr2 = arr1;
        arr2[0] = 99;

        Console.WriteLine("بعد Shallow Copy:");
        Console.WriteLine("arr1[0] = " + arr1[0]); // هيطلع 99 لأن الاتنين بيشاوروا على نفس المكان
        Console.WriteLine("arr2[0] = " + arr2[0]);

        int[] arr3 = (int[])arr1.Clone();

        arr3[0] = 100;

        Console.WriteLine("\nبعد Deep Copy:");
        Console.WriteLine("arr1[0] = " + arr1[0]); // هتفضل 99
        Console.WriteLine("arr3[0] = " + arr3[0]); // هتكون 100
    }
}
```

Question2

**Array.Clone()**

- **تعمل نسخة جديدة** من المصفوفة.
- **للمصفوفات البسيطة** زي int[], double[], لكن لو فيها كائنات **نسخ عميق (Deep Copy)** بتعمل النسخ بيكون **سطحي**(objects) ، **(Shallow)**.
- بتُرجع (object) وبتحتاج تعمل لها **casting** **نسخة من نفس النوع**.

**Array.Copy()**

- بتنسخ البيانات **من مصفوفة لأخرى موجودة بالفعل**.
- لازم تحدد المصفوفتين (المصدر والهدف) وممكن تحدد أماكن النسخ وعدد العناصر.

- مفيدة لو عايز تتحكم في جزء معين من النسخ.

---

Problem3

```
using System;

class Program
{
  static void Main()
  {
    int[,] grades = new int[3, 3];

    for (int student = 0; student < 3; student++)
    {
      Console.WriteLine($"ادخل درجات الطالب رقم {student + 1}:");

      for (int subject = 0; subject < 3; subject++)
      {
        Console.Write($" - المادة رقم {subject + 1}: ");
        grades[student, subject] = int.Parse(Console.ReadLine());
      }

      Console.WriteLine();
    }

    Console.WriteLine("درجات الطلاب:");
```

```
    for (int student = 0; student < 3; student++)
    {
        Console.Write($"الطالب {student + 1}: ");

        for (int subject = 0; subject < 3; subject++)
        {
            Console.Write(grades[student, subject] + " ");
        }

        Console.WriteLine(); // سطر جديد لكل طالب
    }
  }
}
```

Question3:

**Length**

- بتجيب **العدد الكلي لجميع العناصر** في المصفوفة (1 سواء كانت 1D أو 2D أو أكثر).
- ما بتحددش أبعاد، بس بتقولك المصفوفة فيها كام عنصر إجمالًا.

🔷 **GetLength(dimension)**

- بتجيب **عدد العناصر في بُعد معيّن** من أبعاد المصفوفة (البُعد صفر، الأول، الثاني...)
- لازم تمرر رقم البُعد اللي عايز تطلع طوله.

---

Problem4:

```
using System;

class Program
{
```

```
  static void Main()
  {
    int[] numbers = { 5, 2, 9, 1, 7 };
    Console.WriteLine("قبل Sort: " + string.Join(", ", numbers));
    Array.Sort(numbers);
    Console.WriteLine("بعد Sort: " + string.Join(", ", numbers));
    Console.WriteLine();

    Console.WriteLine("قبل Reverse: " + string.Join(", ", numbers));
    Array.Reverse(numbers);
    Console.WriteLine("بعد Reverse: " + string.Join(", ", numbers));
    Console.WriteLine();

    int index = Array.IndexOf(numbers, 5); // بندور على الرقم 5
    Console.WriteLine("مكان الرقم 5 في المصفوفة: " + index);
    Console.WriteLine();

    int[] copy = new int[numbers.Length];
    Array.Copy(numbers, copy, numbers.Length);
    Console.WriteLine("نسخة من المصفوفة (copy): " + string.Join(", ", copy));
    Console.WriteLine();
    Console.WriteLine("قبل Clear: " + string.Join(", ", copy));
    Array.Clear(copy, 1, 2); // امسح عنصرين من index 1
    Console.WriteLine("بعد Clear (مسح عنصرين من index 1): " + string.Join(", ", copy));
  }
}
```

Problem5

```
using System;
class Program
{
   static void Main()
   {
      int[] numbers = { 10, 20, 30, 40, 50 };
      Console.WriteLine("طباعة العناصر باستخدام for:");
      for (int i = 0; i < numbers.Length; i++)
      {
         Console.WriteLine(numbers[i]);
      }
      Console.WriteLine();
      Console.WriteLine("طباعة العناصر باستخدام foreach:");
      foreach (int num in numbers)
      {
         Console.WriteLine(num);
      }
      Console.WriteLine();
      Console.WriteLine("طباعة العناصر بالعكس باستخدام while:");
      int index = numbers.Length - 1;
      while (index >= 0)
      {
         Console.WriteLine(numbers[index]);
         index--;
      }
```

```
  }
}
```

Question5

**foreach:**

 **بيمنع التعديل على المصفوفة عن طريق المتغير داخل اللوب**

- foreach المتغير داخل هو نسخة من العنصر، مش العنصر نفسه.
- بالتالي **تحمي المصفوفة من التعديل غير المقصود**.

 **بيقلل الأخطاء**

- أو تخرج برّه حدود المصفوفةمفيش خطر تغلط في الفهرس ، (IndexOutOfRangeException).
- أكثر أمانًا في الحلقات اللي بس بنقرأ فيها البيان

---

Problem6

```
using System;
class Program
{
  static void Main()
  {
    int number;
    do
    {
      Console.Write("من فضلك ادخل رقم فردي موجب: ");
      string input = Console.ReadLine();
      bool isValid = int.TryParse(input, out number);
      if (!isValid)
      {
        Console.WriteLine("❌ الإدخال مش رقم صحيح.");
```

```
        }
        else if (number <= 0)
        {
            Console.WriteLine("❌ الرقم لازم يكون موجب.");
        }
        else if (number % 2 == 0)
        {
            Console.WriteLine("❌ الرقم لازم يكون فردي.");
        }

    } while (number <= 0 || number % 2 == 0);

    Console.WriteLine("✅ شكراً! الرقم الصحيح هو: " + number);
  }
}
```

Question6:

يمنع البرنامج من إنه ينهار بسبب إدخال غلط.

يحمي من الهجمات والاختراقات.

يضمن إن البيانات اللي داخلة منطقية وصحيحة.

يحسّن تجربة المستخدم برسائل واضحة.

يساعد المبرمج يكتشف الأخطاء بسهولة.

Problem7

```
using System;
class Program
{
   static void Main()
   {
      int[,] matrix = {
         { 1, 2, 3 },
         { 4, 5, 6 },
         { 7, 8, 9 }
      };
      Console.WriteLine("Matrix Elements:");
      for (int row = 0; row < matrix.GetLength(0); row++) // الصفوف
      {
         for (int col = 0; col < matrix.GetLength(1); col++) // الأعمدة
         {
            Console.Write(matrix[row, col] + "\t"); // \t عشان تكون بشكل جدول
         }
         Console.WriteLine(); // ينزل سطر جديد بعد كل صف
      }
   }
}
```

Question7

بشكل أوضح باستخدام (2D array) تقدر تنسّق طباعة المصفوفة الثنائية:

1. **Tab (\t)**: لفصل العناصر داخل كل صف.
2. **Console.Write + Console.WriteLine**: لطباعة الأعمدة والصفوف بشكل منظم.

3. **Padding** أو **التنسيق (String.Format,** أو **Interpolation)**: لمحاذاة الأرقام.

---

Problem8

```
using System;


class Program
{
   static void Main()
   {
      Console.Write("Enter a month number (1-12): ");
      string input = Console.ReadLine();
      int monthNumber;


      if (int.TryParse(input, out monthNumber))
      {


         Console.WriteLine("\nUsing if-else:");
         if (monthNumber == 1)
            Console.WriteLine("January");
         else if (monthNumber == 2)
            Console.WriteLine("February");
         else if (monthNumber == 3)
            Console.WriteLine("March");
         else if (monthNumber == 4)
            Console.WriteLine("April");
         else if (monthNumber == 5)
```

```
    Console.WriteLine("May");
else if (monthNumber == 6)
    Console.WriteLine("June");
else if (monthNumber == 7)
    Console.WriteLine("July");
else if (monthNumber == 8)
    Console.WriteLine("August");
else if (monthNumber == 9)
    Console.WriteLine("September");
else if (monthNumber == 10)
    Console.WriteLine("October");
else if (monthNumber == 11)
    Console.WriteLine("November");
else if (monthNumber == 12)
    Console.WriteLine("December");
else
    Console.WriteLine("Invalid month number.");

Console.WriteLine("\nUsing switch:");
switch (monthNumber)
{
    case 1: Console.WriteLine("January"); break;
    case 2: Console.WriteLine("February"); break;
    case 3: Console.WriteLine("March"); break;
    case 4: Console.WriteLine("April"); break;
    case 5: Console.WriteLine("May"); break;
```

```
            case 6: Console.WriteLine("June"); break;
            case 7: Console.WriteLine("July"); break;
            case 8: Console.WriteLine("August"); break;
            case 9: Console.WriteLine("September"); break;
            case 10: Console.WriteLine("October"); break;
            case 11: Console.WriteLine("November"); break;
            case 12: Console.WriteLine("December"); break;
            default: Console.WriteLine("Invalid month number."); break;
        }
    }
    else
    {
        Console.WriteLine("Invalid input. Please enter a number.");
    }
  }
}
```

Question8

لما يكون عندك **عدد ثابت من القيم المحتملة** (زي أرقام أو رموز ثابتة) وعايز تختار من بينهم بناءً على **قيمة واحدة**. **switch** بدل **if-else** ستخدم

---

Problem9

```
using System;


class Program
{
  static void Main()
```

```
  {

    int[] numbers = { 10, 5, 20, 15, 5, 30, 20 };

    Console.WriteLine("Original Array:");
    foreach (int num in numbers)
    {
      Console.Write(num + " ");
    }

    Array.Sort(numbers);

    Console.WriteLine("\n\nSorted Array:");
    foreach (int num in numbers)
    {
      Console.Write(num + " ");
    }

    int valueToFind = 20;
    int firstIndex = Array.IndexOf(numbers, valueToFind);
    int lastIndex = Array.LastIndexOf(numbers, valueToFind);

    Console.WriteLine($"\n\nFirst index of {valueToFind}: {firstIndex}");
    Console.WriteLine($"Last index of {valueToFind}: {lastIndex}");
  }
}
```

Question9:

**الوقت المُستغرق (Time Complexity) لـ Array.Sort() في C# هو:**

- **O(n log n)** في الحالة المتوسطة وأفضل حالة.
- يستخدم خوارزميات محسنة (مثل C# لأن **O(n log n)** في أسوأ حالة، الأداء يظل غالبًا **IntroSort**).

---

Problem10

```
using System;


class Program
{
   static void Main()
   {
  int[] numbers = { 10, 20, 30, 40, 50 };


      int sumFor = 0;
      for (int i = 0; i < numbers.Length; i++)
      {
         sumFor += numbers[i];
      }
      Console.WriteLine("Sum using for loop: " + sumFor);


      int sumForeach = 0;
      foreach (int num in numbers)
      {
         sumForeach += num;
      }
      Console.WriteLine("Sum using foreach loop: " + sumForeach);
```

```
    }
}
```

Question10:

عند حساب مجموع عناصر المصفوفة، foreachتعتبر غالبًا **أكثر كفاءة** من **for loop**
لأنها:

- foreach (زي إنشاء Enumerator داخليًا) تتجنب بعض العمليات الإضافية التي يقوم بها.
- مما يتيح تحسينات بسيطة في الأداء(index) تعطي تحكم أكبر في الفهرسة ،.

---

Wafaa Mohammed • أنت
–
الآن •

🧠 "ليه تكتب نفس السطر 100 مرة؟ لما ممكن تخلي الكود يشتغل لوحده؟"
لو إنت بدأت تتعلم البرمجة، أكيد سألت نفسك:
"إزاي أخلي الكود يعمل نفس الحاجة كذا مرة من غير ما أكتبها كل مرة؟"
يعني مثلًا عايز تطبع أسماء الطلبة، أو تحسب درجات، أو تمشي على عناصر في List... هتفضل تكتب كل حاجة بإيدك؟
لا طبعًا... هنا ييجي السحر الحقيقي في البرمجة، واسمه: Loops – جمل التكرار ✨

🔁 يعني إيه Loop؟
الـ Loop في C# هي طريقة تخليك تكرر تنفيذ جزء من الكود كذا مرة، من غير ما تعيد وتزيد.
ودي حاجة أساسية بتتعلمها بدري جدًا في أي لغة برمجة.

💡 أنواع Loops في C#

for loop – ✅
لما تكون عارف هتكرر كام مرة
مثالي لو عايز تمشي من 1 لـ 10 مثلًا. بتحط شرط البداية والنهاية وخلاص.

foreach loop – ✅
لما عايز تمشي جوه List أو Array
بتسهل عليك تلف على كل العناصر واحدة واحدة من غير ما تهتم بالفهرس (index).

while loop – ✅
لما الشرط هو اللي بيتحكم في التكرار
بتشتغل طول ما فيه شرط معين بيتحقق. مفيدة جدًا لو مش عارف العدد بالظبط.

do-while – ✅
بس بتشتغل مرة واحدة على الأقل نفس while
يعني الكود هيتنفذ الأول حتى لو الشرط مش متحقق.

⚠️ خد بالك من:
لو نسيت تحدث الفهرس أو الشرط، ممكن تدخل في infinite loop (يعني الكود يفضل شغال ومابيخلصش).

ممكن تستخدم break عشان تطلع برا اللوب في وقت معين.

أو continue عشان تتخطى لفة وتكمل الباقي.

🎯 ليه ده مهم ليك كمبرمج؟
الـ Loops بتوفر وقت، مجهود، وبتخلي الكود أنضف وأسهل في القراءة.
وده جزء مهم من إنك تكون بتكتب كود احترافي وقابل للتطوير.

#CSharp #Loops #برمجة_بالعربي #DotNet
#LearnToCode #ProgrammingTips #كود_ذكي
#SoftwareDevelopment

2

```
using System;

class Program
{

  enum DayOfWeek
  {
    Monday = 1,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday,
    Sunday
  }

  static void Main()
  {
    Console.Write("Enter a number from 1 to 7: ");
    string input = Console.ReadLine();

    if (int.TryParse(input, out int dayNumber))
    {
      if (dayNumber >= 1 && dayNumber <= 7)
      {
```

```
            string dayName = Enum.GetName(typeof(DayOfWeek), dayNumber);


            Console.WriteLine("The day is: " + dayName);
        }
        else
        {
            Console.WriteLine("Number must be between 1 and 7.");
        }
    }
    else
    {
        Console.WriteLine("Invalid input. Please enter a number.");
    }
  }
```

3

```
if (dayNumber >= 1 && dayNumber <= 7)
```


٢:٤٦
%٧٩
Wafaa Mohammed • انت
الآن •
🟩 أول ما بتبدأ برمجة، دايمًا بنسمع "استخدم Array!"
طبيعي جدًا... لأنها من أبسط أنواع الـ data structures
بس الحقيقة؟ إنها مش دايمًا الخيار المثالي، وممكن تقع في مشاكل لو استخدمتها غلط.
تعالى نمشيها واحدة واحدة ونعرف الحكاية...
✅ إمتى تستخدم الـ Array وتكسب؟
عدد العناصر ثابت:
زي لما يكون عندك 7 أيام في الأسبوع، أو 12 شهر، أو درجات 3 مواد.
مش هتزود ولا تنقص، فـ Array هنا perfect!
أداء عالي وسريع:
لو البرنامج محتاج يشتغل بسرعة ومفيش وقت للتفكير – الـ Array خفيفة على السيستم وسريعة في التنفيذ.
استهلاك قليل للذاكرة:
لأنها بتتخزن جنب بعض في الذاكرة (contiguous)،
فلو شغّال في مشروع كبير أو على جهاز ضعيف؟ Array تكسب 💪
الوصول للعناصر مباشر وسهل:
عايز العنصر رقم 3؟ ببساطة تكتب arr[2] وخلاص.
❌ إمتى تقول لا لـ Array؟
عدد العناصر مش معروف أو بيتغير:
زي لما يكون عندك لستة طلبات بتدخل طول الوقت – مش هتعرف تظبط Array حجمها من الأول.
عايز تضيف أو تحذف عناصر:
بتتعب جدًا في الموضوع ده، لازم تعمل نسخة جديدة وتنقل Array كل حاجة تاني 😩
محتاج دوال جاهزة للفلترة – البحث – التعديل:
الـ List<T> أو الـ Dictionary<TKey, TValue> هيسهلوك الحياة أكتر.
مش عايز تتعامل مع فهارس (Indexes):
دايمًا محتاجة منك تبقى فاكر ترتيب كل عنصر، بعكس الـ Array اللي بتخبي التفاصيل دي عنك Collections.
💡 الـ Array عاملة زي علبة مقاسها ثابت...
لو اشتريت لبس جديد ومش لاقي مكان تحطه؟ لازم تغير العلبة 😅
لكن الـ List عاملة زي شنطة فيها سحاب بيوسع... تحط، تشيل، ترتّب، براحتك!
#JuniorDeveloper #CodingForBeginners #LearnToCode
إعجاب
تعليق
إعادة نشر
إرسال
إضافة تعليق...

Part03 bouns

**منظم وسريع، بس محدود – Stack أولًا: الـ**

- الـ Stack هو المكان اللي بيتخزن فيه المتغيرات المؤقتة (زي متغيرات الدوال).
- حجمه ... غالبًا بيكون **مش كبير أوي**:
  - لكل Thread **إلى 4 ميجا** **1**
- عشان كل Thread بياخد Stack المساحة على نحافظ لازم فلازم بيه، خاص.

طيب إيه اللي يحصل لو استهلكت أكتر من كده؟
هتاخد **StackOverflowException** – لو نسيت تغلق recursion أو عملت دالة تستدعي نفسها كتير خطأ مشهور.

---

**كبير بس مش سريع زيه – Heap ثانيًا: الـ**

- الـ Heap هو المكان اللي بيتخزن فيه الكائنات (objects) والمتغيرات اللي ليها عمر أطول.
- بيكبر على قد ما تحتاج – ملوش حجم ثابت.
  - ممكن يوصل لـ **جيجات كتير** طالما فيه RAM متاحة.
- بيتنظف أوتوماتيك عن طريق الـ **Garbage Collector**

---

**طب إيه اللي بيأثر على حجمهم؟** ⚙️

- نظام التشغيل (Windows / Linux)
- نوع المعمارية (32-bit أو 64-bit)
- عدد الـ Threads في البرنامج
- نوع التطبيق (Console, Web, Desktop)
- إعدادات المشروع أو تخصيصك ليها يدويًا

---

- الـ Stack سريع بس محدود.
- الـ Heap كبير بس فيه عبء تنظيف وتأخير شوية.
- اختيارك يعتمد على نوع البيانات، مدة استخدامها، وسيناريو البرنامج

**Time Complexity؟ ليه مهم تفهم**

:علشان لما تيجي تكتب كود أو تختار طريقة لحل مشكلة، تبقى فاهم هل الكود ده

- سريع؟
- ينفع يشتغل على كميات داتا كبيرة؟
- ولا هيقف ويهنج لما العدد يكبر؟

.زي ما بتقارن بين عربية 4 سلندر و8 سلندر، إحنا كمان بنقارن بين الأكواد من ناحية السرعة والكفاءة

---

**:مثال**

:لو قلتلك عدلي الناس اللي واقفين في طابور

- .لو الطابور فيه 5 بس؟ هتخلص بسرعة
- .طب لو فيه 500؟ أكيد هتاخد وقت أطول

.Time Complexity ده كده بيقيس حاجة اسمها

"لكن لو سؤالك هو "مين أول واحد في الطابور؟
.هتجاوب في ثانية، مهما كان العدد
.هنا الوقت ثابت. بنسمي ده إن التعقيد زيرو أو ثابت

---

**:في الكود بقى**

- تمشي على العناصر واحد واحد؟ loop لو بتعمل
  .ده وقت بيزيد مع حجم البيانات
- لو بتنفذ عملية سريعة جدًا مرة واحدة؟
  .ده وقت ثابت، مبيزدش مهما زادت الداتا

---

**Time Complexity** :هو طريقة بنقيس بيها

"الطريقة اللي كتبنا بيها الكود هتشتغل بكفاءة ولا لأ لما نكبر حجم البيانات؟"

.وده مهم جدًا لأي مبرمج لأنه بيفرق بين كود شغال "آه بس بطّيء"، وكود شغال بسرعة مهما كانت الداتا