Linkedin Article - Loop Statements C#

الـ Loop دي حاجة بسيطة جداً - تخيل إنك عايز تعمل حاجة معينة مرات كتير، ف اكيد انك مش هتفضل تكررها بإيدك! ف انت بتخلي الكود هيعملهالك.

أنواع الـ Loops في C#:

1. For Loop

```
for (int i = 0; i < 5; i++)
{
    Console.WriteLine($"رقم مرة {  + 1}");
}
```

يعني إيه؟ هنبدأ من 0، ونفضل نزود 1 كل مرة، لحد ما نوصل 5.

2. While Loop

```
int counter = 0;
while (counter < 3)
{
    Console.WriteLine("Hello!");
    counter++;
}
```

الفكرة: طالما الـ counter أقل من 3، اطبع "Hello!" وزود الرقم.

3. Do-While

```
int number;
do
{
    Console.WriteLine("ادخل رقم أكبر من 10:");
    number = int.Parse(Console.ReadLine());
} while (number <= 10);
```

الحلو فيها: إنها هتشتغل مرة واحدة على الأقل، حتى لو الشرط غلط.

4. Foreach

```
string[] names = {"محمد" ,"فاطمة" ,"أحمد"};
foreach (string name in names)
{
    Console.WriteLine($"أهلاً {name}");
}
```

أسهل حاجة لما تكون عايز تمشي على كل عنصر في مجموعة.

Stack vs Heap Memory

الـ Memory في الكمبيوتر فيه مكانين مختلفين:

الـ Stack

وده عشان تفهمو اكتر خلينا نتخيل مجموعة كتب فوق بعضها - تحط كتاب فوق التاني، وعشان تاخد كتاب لازم تاخد اللي فوق الأول.

إيه اللي بيتحط في الـ Stack؟

```
int age = 25;           // رقم - حجم ثابت
bool isStudent = true;  // true/false - حجم ثابت
char grade = 'A';       // حرف واحد - حجم ثابت
```

ليه دي في الـ Stack؟

- ✧ **حجمها ثابت** - الـ int دايماً 4 bytes، الـ bool دايماً byte 1
- ✧ **الكمبيوتر يعرف هيحجز كام مكان** من البداية
- ✧ **سريع جداً** في الوصول ليها

الـ Heap

الـ Heap ده زي **أوضة فاضية كبيرة** - تحط فيها أي حاجة في أي مكان، وممكن تغير حجمها.

إيه اللي بيتحط في الـ Heap؟

```
string name = "أحمد محمد";        // نص - مش عارفين هيكون كام حرف
List<int> numbers = new List<int>(); // قائمة - ممكن تكبر وتصغر
Person student = new Person();       // object - حجمه متغير
```

ليه دي في الـ Heap؟

- ✧ **حجمها متغير** - الـ string ممكن يكون 5 حروف أو 1000 حرف
- ✧ **مش عارفين هنحتاج كام مكان** من الأول
- ✧ **ممكن تكبر وتصغر** وإحنا شغالين

مثال يوضح الفكرة:

```
// في الـ Stack - سريع ومرتب
int studentCount = 30;      // Stack: 4 bytes ثابتة
bool isActive = true;       // Stack: 1 byte ثابت

// في الـ Heap - مرن بس أبطأ شوية
string studentName = "فاطمة أحمد علي";  // Heap: مش عارفين الحجم
List<string> subjects = new List<string>(); // Heap: ممكن نزود مواد
```

When to use Arrays and when not to?

When to use Arrays?

1. لما العدد ثابت ومعروف

```
// أيام الأسبوع - دايماً 7 أيام
string[] weekDays = {"السبت","الأحد" ,"الاثنين" ,"الثلاثاء" ,"الأربعاء" ,"الخميس" ,"الجمعة"};

// شهور السنة - دايماً 12 شهر
string[] months = new string[12];
```

2. لما تشتغل مع البيانات بالترتيب

```
// درجات الطلاب في الامتحان
int[] examScores = {85, 92, 78, 95, 88};

// حساب المتوسط
int total = 0;
for (int i = 0; i < examScores.Length; i++)
{
    total += examScores[i];
}
double average = total / examScores.Length;
```

3. لما تحتاج سرعة في الوصول للبيانات

```
// الوصول للعنصر بالرقم سريع جداً - O(1)
int firstScore = examScores[0];     // أول درجة
int lastScore = examScores[4];      // آخر درجة
```

4. مع الحسابات الرياضية والمصفوفات

```
// مصفوفة ثنائية للإحداثيات
int[,] gameBoard = new int[8, 8];   // 8 رقعة شطرنج x8
```

When not to use Arrays?

1. لما العدد مش معروف أو بيتغير

```
// مش عارف كام واحد هيسجل - ❌ غلط
int[] registeredUsers = new int[???];

// استخدم - ✅ صح List
List<string> registeredUsers = new List<string>();
```

2. لما تحتاج تضيف أو تمسح كتير

```
// ❌ صعب مع الـ Array
// لازم تعمل array جديد كل مرة

// ✅ سهل مع الـ List
List<string> shoppingCart = new List<string>();
shoppingCart.Add("لبن");           // إضافة
shoppingCart.Remove("خبز");        // حذف
```

3. لما تدور على حاجة معينة كتير

```
// ❌ لازم تدور بنفسك مع الـ Array
bool found = false;
for (int i = 0; i < students.Length; i++)
{
    if (students[i] == "أحمد")
    {
        found = true;
        break;
    }
}

// ✅ أسهل مع الـ List
bool found = studentsList.Contains("أحمد");
```

For vs Foreach

For Loop

```
for (int i = 0; i < array.Length; i++)
{
    // إنت اللي بتتحكم في كل حاجة
    Console.WriteLine(array[i]);
}
```

الفكرة: إنت بتقول للكود "ابدأ من هنا، امشي كده، وقف هنا"

Foreach Loop

```
foreach (string item in array)
{
    // الكود بيمشي لوحده على كل عنصر
```

```
    Console.WriteLine(item);
}
```

الفكرة: إنت بتقول "يلا امشي على كل حاجة واحدة واحدة"

When to use For?

1. لما تحتاج الـ Index

```
string[] students = {"عائشة" ,"محمد" ,"فاطمة" ,"أحمد"};

for (int i = 0; i < students.Length; i++)
{
    Console.WriteLine($"الطالب رقم {i + 1}: {students[i]}");
}
// Output: الطالب رقم 1: أحمد
```

When to use Foreach?

1. لما تقرأ العناصر بس (مش بتعدل)

```
List<string> fruits = {"موز" ,"برتقال" ,"تفاح"};

foreach (string fruit in fruits)
{
    Console.WriteLine($"أحب {fruit}");  // قراءة بس ✅
}
```

Nested For vs Single For

الـ Nested For ده **for loop جوا for loop تاني** - زي البصلة، طبقة جوا طبقة!

Big O Notation:

Single For Loop - O(n)

```
// خطي n هيشتغل – مرة
for (int i = 0; i < n; i++)
{
    Console.WriteLine(i);
}
// لو n = 1000 ← 1000 عملية
```

Nested For Loop - O(n²)

```
// تربيعي n × n هيشتغل – مرة
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        Console.WriteLine($"{i}, {j}");
    }
}
// لو n = 1000 ← 1,000,000 عملية! 😱
```

When Nested For is justified?

1. الـ 2D Arrays

```
int[,] matrix = new int[100, 100];
```

```
// لازم nested - مافيش بديل
for (int row = 0; row < 100; row++)
{
    for (int col = 0; col < 100; col++)
    {
        matrix[row, col] = row + col;
    }
}
```

2. مقارنة كل عنصر بكل عنصر

```
string[] students = {"عائشة" ,"محمد" ,"فاطمة" ,"أحمد"};

// البحث عن الأسماء المتشابهة
for (int i = 0; i < students.Length; i++)
{
    for (int j = i + 1; j < students.Length; j++)
    {
        if (students[i].StartsWith(students[j].Substring(0, 1)))
        {
            Console.WriteLine($"{students[i]} و {students[j]} يبدآن بنفس الحرف");
        }
    }
}
```

إزاي نحسن أداء الـ Nested For:

1. Early Break

```
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        if (someCondition)
        {
            break; // وقف الـ inner loop
        }
        // أو
        if (anotherCondition)
        {
            goto exitBothLoops; // وقف الكل
        }
    }
}
exitBothLoops:
```

2. تقليل الحدود

```
// يشتغل كامل inner loop بدل ما تخلي الـ
for (int i = 0; i < n; i++)
{
    // بدل من 0 ابدأ من i+1
    for (int j = i + 1; j < n; j++)
    {
        // دي تقلل العمليات للنص!
        CompareItems(i, j);
    }
}
```

Default Stack and Heap sizes

Default sizes:

Stack Size

✧ **Windows 32-bit:** 1 MB (1,048,576 bytes)

- ✧ **Windows 64-bit:** 1 MB (1,048,576 bytes)
- ✧ **Linux:** 8 MB (8,388,608 bytes)
- ✧ **macOS:** 8 MB (8,388,608 bytes)

Heap Size الافتراضي:

- ✧ **يبدأ صغير:** حوالي 4 MB
- ✧ **يكبر حسب الحاجة** - مافيهوش حد أقصى محدد
- ✧ **الحد الأقصى:** RAM المتاح في النظام
- ✧ الـ **Garbage Collector** بيظبط المساحة

إيه معنى الأرقام دي عملياً؟

Stack - 1 MB

```
// كل method call بياخد مساحة في الـ Stack
void RecursiveMethod(int n)
{
    int localVar = n;        // 4 bytes
    bool flag = true;        // 1 byte
    double price = 25.5;     // 8 bytes

    if (n > 0)
    {
        RecursiveMethod(n - 1); // استدعاء جديد = مساحة جديدة
    }
}

// 50~ لو كل استدعاء ياخد bytes
// 1 MB ÷ 50 = استدعاء 20,000 حوالي recursive
```

Heap - بيكبر إزاي؟

```
// الـ Heap بيكبر تلقائياً
List<string> bigList = new List<string>();

// كل string جديد بيروح الـ Heap
for (int i = 0; i < 1000000; i++)
{
    bigList.Add($"عنصر رقم {i}"); // الـ Heap بيكبر لوحده
}
// ممكن يوصل لـ GB حسب الـ RAM المتاح
```

What is time complexity

الـ Time Complexity ده ببساطة هو **مقياس لسرعة الكود بتاعك** - مش بالثانية والدقيقة، لأ، بنقيس هو هياخد وقت قد إيه لما حجم البيانات يكبر.

يعني لو عندك كود بيشتغل على 10 عناصر، أكيد هيكون سريع. طب لو اشتغل على 10 مليون عنصر؟ هل هيفضل سريع ولا هيموت؟ الـ Time Complexity هو اللي بيجاوب على السؤال ده.

أشهر أنواع الـ Time Complexity:

1. O(1) - Constant Time

ده **أسرع حاجة ممكنة**. يعني الكود بياخد نفس الوقت بالظبط، سواء شغال على عنصر واحد أو مليون عنصر.

```
// هاتلي أول عنصر في الـ array
string[] names = {"أحمد", "فاطمة", "محمد"};
string firstName = names[0]; // دي دايماً سريعة بنفس الدرجة - O(1)
```

الفكرة: عملية واحدة مباشرة مبتعتمدش على حجم البيانات.

2. O(n) - Linear Time

هنا الوقت بيزيد **بشكل خطي** مع زيادة حجم البيانات (n). لو البيانات زادت للضعف، الوقت هيزيد للضعف.

```
// اطبع كل الأسماء
string[] names = {"محمد" ,"فاطمة" ,"أحمد"}; // n = 3
foreach (string name in names) // هيلف 3 مرات - O(n)
{
    Console.WriteLine(name);
}
```

الفكرة: زي الـ Single For Loop، بتمشي على كل العناصر مرة واحدة.

3. O(n²) - Quadratic Time

ده **بطيء جداً** لما البيانات تكبر. الوقت بيزيد بشكل تربيعي. لو البيانات زادت 10 مرات، الوقت هيزيد 100 مرة!

```
// قارن كل اسم بكل اسم
for (int i = 0; i < n; i++) // هيلف n مرة
{
    for (int j = 0; j < n; j++) // هيلف n كمان مرة
    {
        // العملية دي هتتنفذ n * n مرة
    }
}
```

الفكرة: زي الـ Nested For Loop، كل عنصر بيتقارن بكل العناصر التانية.

4. O(log n) - Logarithmic Time

ده **سريع جداً** ويعتبر مثالي للبيانات الكبيرة. الوقت بيزيد ببطء شديد جداً مع زيادة حجم البيانات.

```
// البحث في array مترتب (Binary Search) مثال:
// مش بتمشي على كل العناصر
// كل مرة بالنص وتدور في النص الصح array بتقسم الـ
// لو عندك مليون عنصر، هتحتاج حوالي 20 خطوة بس عشان تلاقي اللي بتدور عليه
```

الفكرة: زي ما بتدور على كلمة في القاموس، مش بتفتح صفحة صفحة، لأ، بتفتح من النص وتقرر تكمل في أنهي نص.