جامعة البعث
الكلية التطبيقية
قسم تقنيات الحاسوب

برمجة متقدمة
السنة الثانية
عملي
محاضرة ثانية

م. زكي نقولا م. بتول الليوس م. سارة معروف

مفهوم الدالة Method في C#

هي عبارة الكتلة block من التعليمات برمجية تحتوي على سلسلة من تعليمات برمجية Instructions هذه التعليمات تقوم بتنفيذ مهمة محددة. والـ Method تنفذ عن طريق الإستدعاء عند الحاجة فقط وفي حال لم يتم الإستدعاء فإن كتلة التعليمات هذه لاتنفذ. **الـ Methods في سي شارب** يمكن أن تكون :

- **دوال ترجع قيمة (Value-Returning Methods):** تعتبر هذه الدوال من أهم أنواع الدوال في C# تقوم بتنفيذ سلسلة من الأوامر وترجع قيمة بعد الانتهاء. يمكن استخدام دوال الـ Value-Returning لتنفيذ حسابات معقدة واسترجاع النتائج للاستخدام في أماكن أخرى من البرنامج.
- **دوال الإجراء (Void Methods):** تعتبر هذه الدوال من أكثر الأنواع شيوعًا في C# . تقوم بتنفيذ سلسلة من الأوامر ولا ترجع قيمة. يتم استخدام دوال الإجراء لتنفيذ أعمال محددة دون الحاجة إلى إرجاع قيمة محددة..

كما يمكن أن تكون الـ Method جاهزة للإستدعاء وموجودة في المكتبات الخاصة بـ .Net.

إنشاء Methods في C#

لإنشاء دالة Method في C# ، يتم تعريفها بشكل مستقل داخل Class ويتم تحديد اسم الدالة Method Name وقائمة المعاملات Parameters List (إذا كانت موجودة) و (إذا كانت ثابتة)، ثم يتم تنفيذ سلسلة من العبارات داخل الدالة

اذاً **تتكون الـ Method في سي شارب** من:

- اسم Method Name و الاسم معرفًا ذا مغزى يشير إلى الغرض العام المراد من إنشاء الدالة و يتم استدعاء الدالة باسمها.(عند تسمية الدوال في البرمجة ، يجب أيضًا النظر في الشفافية وإعطاء المعنى للعمليات. فتكون تسمية الدوال بطريقة واضحة تعكس العملية التي تقوم بها. هذا يسهل فهم البرنامج وتوثيقه، ويمكن لفريق البرمجة تبادل المعرفة بشكل أفضل حيث العمل الجماعي ويساهم في الانسيابية والتفاعل بين المبرمجين).
- وجسم Method Body ونعني بالجسم الدالة مجموعة التعليمات التي سيتم تشغيلها عند استدعاء الدالة.
- ويمكن إعطاء الـ Methods قائمة المعاملات Parameters List وهي البيانات المراد معالجتها بواسطة تعليمات الدالة (اختياري على حسب ما يتطلب).
- نوع القيمة المُرجَعة Return Type (اذا كانت الدالة من النوع الذي يرجع قيمة).

دوال ترجع قيمة (Value-Returning Methods)

طريقة الإعلان عن دالة ترجع قيمة Methods في C# , تكون الصيغة العامة **للإعلان عن دالة ترجع قيمة في لغة سي شارب** كالتالي :

```
returnType MethodName( parametersList )

{

// Method body statements go here
```

```
return value;

}
```

مثال:

```
double Square (double a){
 Return a*a;
}
int Add (int a, int b){
 return a+b;
}
double num = Square (3);
Console.WriteLine(num);
Console.WriteLine(Add(2,3));
```

دوال الإجراء(Void Methods)

طريقة الإعلان عن دالة Method لا ترجع قيمة. تكون الصيغة العامة **للإعلان عن دالة Method لاترجع قية في لغة سي شارب** كتالي:

```
void MethodName (parametersList)

{

// Method body statements go here

}
```

نرى هنا اننا نستخدم الكلمة المحجوزة **void**في بيان الإعلان عنها متبوع باسم الدالة كذلك تستخدم و الأقواس و Parameters Listكما ذكرنا في مع الدالة التي ترجع قيمة.

```
void Square (double a){
Console.WriteLine(a*a);
}
void Add (int a, int b){
```

```
 Console.WriteLine(a+b);
}
Square (3);
Add(2,3);
```

ملاحظة : عند استدعاء أي دالة تستقبل Parameters لابد ان يكون نوع البيانات المرسلة (arguments) عند الاستدعاء مطابقة مع Parameter List من حيث النوع والترتيب.

استخدام الدوال Methods من Net.

عند استدعاء الدالة في البرنامج، يتم تمرير المعاملات اللازمة إليها (إن وجدت)، وستقوم الدالة بتنفيذ العمليات المحددة وتُرجِع النتيجة إذا كانت تحتاج إلى ذلك. بالتالي، يتم **استخدام الدوال في C# ل**تنفيذ المهام المتكررة والعمليات المعقدة والتلاعب بالبيانات.ويمكن أن تكون الـ Method جاهزة للإستدعاء وموجودة في المكتبات الخاصة بـ Net.ذكرنا هذا بإيجاز في مقال مقدمة إلى سي شارب وفيما يلي شرح لدوال الدوت نت:

- يوجد في Net.مايُعرف بـ NET Framework Class Library وهو مجموعة من المكتبات تسمى Namespace Collections معرفة مسبقاً تستخدم في تطبيقات Net.أي لكل Namespace اسم تستدعى بواسطة باستخدام الكلمة المحجوزة "**using**".
- وتحتوي الـ Namespace الواحدة على مجموعة من الـ Classes حيث يحتوي كل Class من هذه الـ Classes على مجموعة من المتغيرات والدوال الـ Method الجاهزة للإستخدام.
- لكي يتسنى لنا العمل مع الدوال الـ Methods الجاهزة لابد من تضمين الـ Namespace الخاصة بها في البرنامج كما يمكن تحديد الـ Class الخاص بها من هذه الـ Namespace حيث تستخدم الكلمة المحجوزة using لتضمين الـ Namespace ويستخدم المعامل (.) مع الـ Class او لتحديد الدالة المراد استدعائها لنأخذ أمثلة ليتضح الأمر:
    - عندما نكتب:

- using System.IO;

هذا يعني استخدم الـ Namespace التي اسمها "System" والـ Class الذي اسمه "IO" الموجود في System.

وعند استدعاء الدوال نكتب اسم الـ Class أولاً ثم اسم الدالة مثلاً:

```
Console.WriteLine();
```

الـ Class هو "Console " والدالة هي ()WriteLine

أو

```
Convert.ToDouble();
```

- الـ Class هو "Convert" والدالة هي()ToDouble

وتوفر System Namespace مجموعة من Classes المهمة منها **Math Class**والذي يحتوي على عدد من الدوال التي تقوم بأداء العمليات الحسابية المعروفة

مما سبق نستنتج أن يمكن إعادة استخدام **الدوال Method** المعرفة مسبقاً في البرامج كوحدات بناء لإنشاء تطبيقات جديدة وهذا يفيد في:

- تجنب تكرار التعليمات البرمجية.
- يؤدي تقسيم التطبيق إلى Methods إلى تسهيل التطبيق وصيانته.

لهذا تُعد **الدوال Methods من الأدوات الرئيسية في لغة C# فهي تسهم بشكل فعال في** تنظيم التنفيذ للبرامج. وتُسهم أيضاً في جعل الكود أكثر قابلية للقراءة والصيانة وإعادة الاستخدام.

المصفوفات في لغة C#

يمكن **تعريف المصفوفة Array في لغة C#** انها نوع الـ Data Structure الذي يمثل مجموعة عناصر (Elements) مفهرسة و مترابطة مع بعضها البعض ولها نفس نوع البيانات. و **المصفوفة Array في سي شارب** يمكن أن تكون خطية أو غير خطية حيث تدعم سي شارب أشكال متعددة من المصفوفات Arrays مثل:

- المصفوفة الخطية Linear Array وهي أبسط أشكال المصفوفات تتكون من بُعد واحد
- المصفوفة غير خطية Non Linear Array وهي المصفوفة متعددة الأبعاد.
- الـ ArrayList تطبيق للخصائص القوائم List بواسطة مصفوفة خطية.

تُستخدم المصفوفات Arrays بشكل واسع في البرمجة لتنظيم البيانات والعمليات عليها. توفر لغة سي شارب العديد من الدوال والخصائص المفيدة للتعامل مع المصفوفات بطرق مختلفة.و يجدر بالذكر أنه يمكن استخدام مكتبات لغة سي شارب لتوفير دوال وأدوات إضافية للتعامل مع المصفوفات بشكل أكثر تعقيدًا وتخصيصًا.

المصفوفة الخطية Linear Array في C#

المصفوفة الخطية Linear Array في سي شارب هي مصفوفة يتم ترتيب البيانات فيها ببعد واحد، أيضاً المصفوفة الخطية Linear Array محدودة العناصر أي أن المصفوفة لها طول ثابت فهي توجد في Block واحد من الذاكرة يُحجزعند الإعلان عنها. ويتم الوصول إلى هذه العناصر بواسطة عنوان الفهرسة .Indexing كما يتم تحديد نوع العناصر للـ Array عند الإعلان عنها فالصيغة العامة للإعلان عن Array تكون كالتالي:

```
ArrayType[] ArrayName = new ArrayType[ Number of array's elements(array Length)];
```

من الصيغة العامة يتضح لنا أن **بيان الإعلان عن المصفوفة Array في لغة سي شارب** يبدأ بتحديد نوع البيانات للمصفوفة ثم الاقواس المربعة [ ] ثم اسم المصفوفة ثم معامل التعيين (=) وبما ان المصفوفة Object نستخدم كلمة **new** لنحدد بعدها نوع العناصر وعددها للمصفوفة حيث أن عدد العناصر هو طول المصفوفة Array Length مثلاً عندما نريد اعلان عن مصفوفة أرقام صحيحة Integer باسم A وعدد عناصر 5 ستكون **طريقة تعريف المصفوفة Array في لغة سي شارب** كالتالي:

```
int [ ] A = new [5];
```

عند الإعلان عن المصفوفة Array بهذه الطريقة سيتم اسناد قيم افتراضية للعناصر المصفوفة حسب نوع البيانات للمصفوفة فتكون القيم الافتراضية للعناصر الرقيمة 0 والمنطقية False و Null لغيرها يمكن كذلك إسناد القيم مباشرة للعناصر المصفوفة عند الإعلان يكون **بيان الإعلان عن المصفوفة** كالتالي:

```
int [ ] A = {7, 3, 0, -2, 1};
```

مثال

```
int [] array ;
array = new int [5];
Console.WriteLine("{0}  {1,8}" , "index","value")
for(int i=0 ; I < array.Lenght; i++)
  Console.WriteLine("{0,5}    {1,8}", I , array[i]);
```

الـ Output للمثال:

index	value
0	0
1	0
2	0
3	0
4	0

في المثال رأينا طريقة المرور على عناصر المصفوفة وطباعتها من خلال حلقة التكرار **for** loop كما استخدمت الخاصية **length** لتمثيل طول المصفوفة في الـ loop. نلاحظ أيضاً ترتيب العناصر يبدأ مع القيمة 0 ، والقيمة الافتراضية للعناصر الرقمية هي 0.

تعليمة foreach و المصفوفات Arrays في C#

مع **المصفوفات Arrays في لغة سي شارب** تستخدم حلقات التكرار Loop للوصول الى عناصر المصفوفة كلها للتعديل أو لطباعتها . وهنالك ايضاً تعليمة التكرار **foreach** التي يمكن أن تستخدم للوصول الى **عناصر المصفوفة Array** دون التعديل عليها فأي محاولة لتعديل قيمة معينة من قيم عناصر المصفوفة بواسطة foreach تعد خطأ برمجي، أي **لا يمكن استخدام تعليمة foreach بدلاً من تعليمة for لإجراء عمليات على عناصر المصفوفة Array** .

و يوضح المثال التالي طريقة استخدام **foreach** مع **المصفوفة Array في سي شارب** :

```
int [] array = { 1, 2, 3};

foreach( int number in array)

  Console.WriteLine(number);
```

من المثال يتضح لنا بناء جملة foreach في لغة سي شارب يكون كتالي :

```
foreach( Type Identifier in ArrayName )
```

- Type Identifier النوع والمعرف هما نوع واسم لمتغير التكرار.
- الكلمة المحجوزة (**in**) تنعي اختبر قيمة متغير التكرار إذا كانت موجودة في المصفوفة Array.
- ArrayName هو اسم المصفوفة Array التي يتم التكرار من خلالها.

في foreach يجب أن يتطابق نوع متغير التكرار مع نوع العناصر في المصفوفة. ويمثل متغير التكرار القيم المتتالية في المصفوفة عبر دورات التكرار المتتالية لعبارة foreach. و تدعم كثير من لغات البرمجة جملة **foreach للعمل مع المصفوفات Arrays**.

المصفوفة Array و الدوال Methods

في سي شارب يمكن تمرير المصفوفات Arrays كا Arguments لمعالجتها بواسطة دالة

قبل البدء بمناقشة تمرير المصفوفات Array لابد أن نشرح الفرق في حالة تمرير قيمة لعنصر من عناصر المصفوفة أو تمرير المصفوفة كاملة. لابد أن نعرف أن أي تغيير على عنصر من عناصر المصفوفة من خلال دالة أو تعليمة سيؤثر على قيمة العنصر فالتعامل مع قيم عناصر المصفوفة يختلف عن التعامل مع المصفوفة كـ Object.

ففي حال تم **تمرير قيمة عنصر معين من عناصر المصفوفة الى دالة** وتمت معالجة قيمة هذا العنصر أو تمرير تمرير مصفوفة كا Argument وتمت معالجة كل العناصر عبر المرور على العناصر عنصر عنصر في المصفوفة هنا سيؤثر التغيير على قيم عناصر المصفوفة في خارج الدالة هذا لأننا نتعامل مع Object فنحن هنا نغير في بيانات هذا الـ Object لاحظ /ي نتائج المثال التالي لتتضح الفكرة:

```
class PassArray
    {
        public static void Main(string[] args)
        {
            int [] array = {1, 2, 3, 4, 5};
            Console.WriteLine("Effects of passing reference to entier array:\n"+
                              "The values of the original array are:" );
            //output original array elements
            foreach (int value in array)
            Console.Write("    {0}", value);

            ModifyArray(array);     //pass array reference
            Console.WriteLine("\n\nThe values of the modified array are:" );
            //output modified array elements
            foreach (int value in array)
            Console.Write("    {0}", value);

            Console.WriteLine("\n\nEffects of passing array element Value:\n"+
                              "array[3] before ModifyElement: {0}", array[3] );
            ModifyElement(array[3]);    // attempt to modify array[3]

            Console.WriteLine("array[3] after ModifyElement: {0}", array[3] );
        }
        public static void ModifyArray( int[] array2)
        {
            for (int counter = 0; counter < array2.Length; counter++)
            array2[counter] *= 2;
        }
        public static void ModifyElement( int element)
        {
            element *= 2;
            Console.WriteLine( "Value of element in ModifyElement: {0}", element);
        }
    }
```

لتكن النتيجة

The values of the original array are:

1 2 3 4 5

The values of the modified array are:

2 4 6 8 10

Effects of passing array element Value:

array[3] before ModifyElement: 8

Value of element in ModifyElement: 16

array[3] after ModifyElement: 8

في المثال السابق رأينا طريقة تمرير المصفوفة الى دالة وتمثل الطريقة المستخدمة أسلوب التمرير بالقيمة Pass by value ففي حال تمرير المصفوفة Array الى دالة Method ايضاً يكون التمرير على نوعين:

تمرير المصفوفة Array بالقيمة Pass by value وهنا ستأخذ الدالة نسختها من المصفوفة كـ Object وفي حال تعديل الدالة على قيم العناصر سيظهر التعديل على عناصر المصفوفة عنصر عنصر سيظهر التغيير على المصفوفة خارج الدالة . هذا لأن المصفوفة Object فالتغير في قيم العناصر عنصر عنصر يعتبر تغيير في بيانات الـ Object.

أما في حال التعديل على كامل المصفوفة والتعامل مع المصفوفة كا متغير هنا سيظهر التأثير على المصفوفة داخل الدالة فقط ولن يظهر على المصفوفة خارج الدالة. لاحظ /ي النتائج في المثال التالي :

```
class ArrayReferenceTest
    {
        public static void Main(string[] args)
        {
            //Careat and initialize arrayValue
            int [] arrayValue = {1, 2, 3};
            //copy the reference in variable arrayValue
            int[] arrayValueCopy = arrayValue;
            Console.WriteLine("Test passing arrayValue by value");
            Console.Write("\ncontents of arrayValue before calling DoubleArray:\n\t");
            //display content of  arrayValue
            foreach(int i in arrayValue)
            Console.Write("{0} ", i);

            //Pass variable arrayValue by value to DoubleArray
            DoubleArray(arrayValue);
            Console.Write("\ncontents of arrayValue after calling DoubleArray:\n\t");
            //display content of  arrayValue
             foreach(int i in arrayValue)
            Console.Write("{0} ", i);

            //test whether reference was changed by DoubleArray
            if ( arrayValue == arrayValueCopy)
            Console.WriteLine("\n\nThe references refers to the same array ");
            else
            Console.WriteLine("\n\nThe references refers to the different arrays ");
        }
        public static void DoubleArray( int[] array)
        {
            //Double each element by value
            for (int i = 0; i < array.Length; i++)
            array[i] *= 2;
            //create new object and assign its reference to array
            array = new int[] {11 , 12, 13};
            Console.Write("\n\nArray in the DoubleArray Method:\n\t");
            foreach(int i in array)
            Console.Write("{0} ", i);

        }
```

لتكن النتيجة

Test passing arrayValue by value

contents of arrayValue before calling FirstDouble:

1 2 3

Array in the DoubleArray Method:

11 12 13

contents of arrayValue after calling DoubleArray:

2 4 6

The references refers to the same array

أيضاً يمكن **تمرير المصفوفة Array بطريقة Pass by reference** . وفي هذه الحالة سيتم استقبال المصفوفة Array مع الإشارة إلى مرجع (Reference) بمعنى الوصول الى موقع المصفوفة الأصلية في الذاكرة واستخدامه داخل الدالة لاحظ *اي* النتائج في المثال التالي ليتضح الأمر:

```
class ArrayReferenceTest
    {
        public static void Main(string[] args)
        {
            //Careat and initialize arrayValue
            int [] arrayValue = {1, 2, 3};
            //copy the reference in variable arrayValue
            int[] arrayValueCopy = arrayValue;
            Console.WriteLine("Test passing arrayValue by value");
            Console.Write("\ncontents of arrayValue before calling DoubleArray:\n\t");
            //display content of  arrayValue
            foreach(int i in arrayValue)
            Console.Write("{0} ", i);

            //Pass variable arrayValue by value to DoubleArray
            DoubleArray(arrayValue);
            Console.Write("\ncontents of arrayValue after calling DoubleArray:\n\t");
            //display content of  arrayValue
             foreach(int i in arrayValue)
            Console.Write("{0} ", i);

            //test whether reference was changed by DoubleArray
            if ( arrayValue == arrayValueCopy)
            Console.WriteLine("\n\nThe references refers to the same array ");
            else
            Console.WriteLine("\n\nThe references refers to the different arrays ");
        }
        public static void DoubleArray( int[] array)
        {
            //Double each element by value
            for (int i = 0; i < array.Length; i++)
            array[i] *= 2;
            //create new object and assign its reference to array
            array = new int[] {11 , 12, 13};
            Console.Write("\n\nArray in the DoubleArray Method:\n\t");
            foreach(int i in array)
            Console.Write("{0} ", i);

        }
```

لتكن النتيجة

Test passing ArrayRef reference by reference

contents of ArrayRef before calling DoubleArray:

1 2 3

Array in the DoubleArray Method:

2 4 6

contents of ArrayRef after calling DoubleArray:

11 12 13

The references refers to the different arrays

ملاحظة :

يعد تمرير المصفوفات Arrays والكائنات Objects حسب المرجع أمرًا منطقيًا لأسباب تتعلق بالأداء. إذا تم تمرير المصفوفات بالقيمة، فسيتم تمرير نسخة من كل عنصر. بالنسبة إلى المصفوفات الأكبر حجمًا والتي يتم تمريرها بشكل متكرر ، قد يؤدي ذلك إلى إضاعة الوقت يستهلك مساحة تخزين كبيرة لنسخ المصفوفات - تؤدي كلتا المشكلتين إلى ضعف الأداء .