Ammar's To-Do List

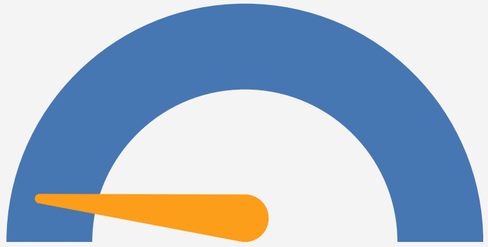
21-02-2025

Dashboard

Daily To-Do List

Weekly To-Do List

Monthly To-Do List



Yearly To-Do List

Notes

صلي علي من زار يهوديا غاب
يوما عن اذيته

Goals Today

	Ternary Operator
	Foreach
	Do-while loop
	Switch Statement
	Methods
	Value Types & Reference Types
	Type Casting

Goals Today

	String Formatting
	String Split & Join
	StringBuilder
	Classes & Objects
	Constructor
	Properties
	Access Modifiers

Goals Today

	Static Classes
	ref & out Keywords
	Enums
	Abstract Classes
	Interfaces
	ArrayList
	Stack & Queue

Motivational Quotes

“We cannot solve problems with the kind of thinking we employed when we came up with them.” —Albert Einstein

“Learn as if you will live forever, live like you will die tomorrow.” —Mahatma Gandhi

“Stay away from those people who try to disparage your ambitions. Small minds will always do that, but great minds will give you a feeling that you can become great too.” —Mark Twain

“When you change your thoughts, remember to also change your world.” —Norman Vincent Peale

Ternary Operator

C# Ternary Operator هو اختصار لجملة if-else وهو مكتوب بالشكل

```
condition ? expression1 : expression2;
```

- condition: الشرط الذي سيتم التحقق منه (يجب أن يكون نوعه bool).
- expression1: القيمة التي سيتم إرجاعها إذا كان الشرط true.
- expression2: القيمة التي سيتم إرجاعها إذا كان الشرط false.

```
int number = 10; string result = (number % 2 == 0) ? "Even" : "Odd";
Console.WriteLine(result);
```

Foreach

foreach: تُستخدم للتكرار عبر العناصر داخل مجموعة (collection) مثل المصفوفات (arrays)، والقوائم (<List<T>)، والمجموعات (<IEnumerable<T>).

الشكل العام لـ foreach

```
foreach (var item in collection)
{
تنفيذ العمليات على كل عنصر //
}
```

```
int[] numbers = { 1, 2, 3, 4, 5 };
foreach (int num in numbers) {
Console.WriteLine(num); }
```

Do-while loop

do-while: تُستخدم لتنفيذ الكود على الأقل مرة واحدة، ثم تكراره طالما أن الشرط true. الفرق بينها وبين while هو أن do-while تنفّذ الكود قبل التحقق من الشرط.

```
int x = 5;


do
{
Console.WriteLine(x);
x--;
} while (x > 0);
```

Switch Statement

C#، switch تُستخدم كبديل لجملة if-else if عندما يكون لديك شرط معين يتحقق من قيمة متغيرة ومقارنتها بعدة قيم محتملة.

```
switch (expression)
{
case value1:
// تنفيذ الكود إذا كانت expression تساوي value1
break;

case value2:
// تنفيذ الكود إذا كانت expression تساوي value2
break;

default:
// تنفيذ الكود إذا لم تتطابق أي من القيم
break;
}
```

```
int score = 85;

switch (score)
{
case int s when s >= 90:
Console.WriteLine("ممتاز!");
break;
case int s when s >= 75:
Console.WriteLine("جيد جدًا!");
break;
case int s when s >= 50:
Console.WriteLine("مقبول!");
break;
default:
Console.WriteLine("راسب!");
break;
}
```

expression: هو المتغير أو التعبير الذي سيتم التحقق منه.

case value:: إذا كان expression يساوي value، يتم تنفيذ الكود في هذا case.

break;: يستخدم للخروج من switch بعد تنفيذ الكود، وإلا سيتم تنفيذ جميع الحالات التالية (يسمى fall-through).

default:: يتم تنفيذه إذا لم يتطابق expression مع أي case.

Methods

الـ Methods تُستخدم لتنفيذ مجموعة من التعليمات عند استدعائها، مما يساعد على إعادة استخدام الكود وتقسيم البرنامج إلى أجزاء صغيرة وسهلة الفهم.

```
returnType MethodName(parameters)
{
// الكود الذي سيتم تنفيذه
return value; // (اختياري)
}
```

- returnType: نوع البيانات التي تُرجعها الدالة (void إذا لم تُرجع قيمة).
- MethodName: اسم الدالة.
- parameters: (اختياري) المتغيرات التي يمكن تمريرها للدالة.
- return: (اختياري) تُستخدم لإرجاع قيمة من الدالة.

دالة بدون معاملات وبدون إرجاع قيمة (void)

```
csharp
CopyEdit
void SayHello()
{
Console.WriteLine("مرحبًا بك في C#!");
}
SayHello();
```

```
int Add(int a, int b) { return a + b; }
;int result = Add(5, 10)
;Console.WriteLine(result)
```

Value Types & Reference Types

في C# بنقسم أنواع البيانات إلى نوعين رئيسيين:

1️⃣ Value Types (أنواع القيم)

2️⃣ Reference Types (أنواع المراجع)

Value Types: تخزن البيانات نفسها مباشرةً في الذاكرة.

Reference Types: تخزن مرجعًا (عنوانًا) إلى البيانات في الذاكرة.

1. Value Types (أنواع القيم)

الأنواع دي تُخزن مباشرةً في الذاكرة (stack)، يعني كل متغير يحمل نسخة منفصلة من القيمة الخاصة به.

✅ أمثلة على Value Types:

الأنواع الأساسية (int, double, char, bool, float, decimal)

2. Reference Types (أنواع المراجع)

الأنواع دي تُخزن في heap والمرجع يُخزن في stack، مما يعني أن المتغير لا يخزن القيمة نفسها، بل يخزن عنوان الذاكرة حيث يتم تخزين البيانات.

✅ أمثلة على Reference Types:

class,interface,stringnarray

Type Casting

Type Casting، C# هو عملية تحويل قيمة من نوع بيانات معين
إلى نوع آخر. تنقسم عملية التحويل إلى نوعين:

1️⃣ Implicit Casting (التحويل الضمني)

2️⃣ Explicit Casting (التحويل الصريح)

1. Implicit Casting (التحويل الضمني)
يحدث تلقائيًا عندما يكون التحويل آمنًا (أي بدون فقدان بيانات).
يتم من نوع أصغر إلى نوع أكبر (مثلاً int إلى double).

2. Explicit Casting (التحويل الصريح)
يتم بشكل يدوي لأنه قد يؤدي إلى فقدان البيانات أو حدوث أخطاء تشغيلية.
يتم من نوع أكبر إلى نوع أصغر (مثلاً double إلى int).

```
string strNumber = "123";
int number = int.Parse(strNumber);

Console.WriteLine(number); // Output: 123
```

```
double myDouble = 9.78;
int myInt = Convert.ToInt32(myDouble); // يتم التقريب لأقرب عدد صحيح

Console.WriteLine(myInt); // Output: 10
```

String Formatting

1. String Concatenation (الدمج باستخدام +)
طريقة تقليدية يتم فيها دمج النصوص باستخدام +.

2. String Interpolation ($"...") [الأفضل]
أسهل وأوضح طريقة لإدراج القيم داخل النصوص باستخدام $.

3.String.Format (String.Format(...))
تستخدم نفس مبدأ interpolation لكن بشكل مختلف.

4. Composite Formatting (Console.WriteLine("
{0} {1}"))

5. StringBuilder (لتحسين الأداء عند التعديل المتكرر)
من الأفضل، عند التعامل مع تعديلات متكررة على النصوص
استخدام StringBuilder لأنه أكثر كفاءة من string.

String Split & Join

يمكن استخدام Split لتقسيم النصوص و Join لإعادة تجميعها في نص واحد.

1. Split() – تقسيم النص إلى مصفوفة
تُستخدم Split() لتقسيم النص إلى مجموعة من الكلمات بناءً على فاصل معين (delimiter).

```
string text = "Hello,World,From,CSharp";
string[] words = text.Split(',');

foreach (string word in words)
{
Console.WriteLine(word);
}
```

2. Join() – دمج النصوص من مصفوفة إلى string
تُستخدم Join() لإعادة تجميع النصوص في سلسلة واحدة مع وضع فاصل (separator) بين كل عنصر.

```
string[] words = { "Hello", "World", "From", "CSharp" };
string result = string.Join(" ", words);

Console.WriteLine(result);
```

Classes & Objects

Classes & Objects الكلاس (Class) هو القالب أو المخطط الذي نستخدمه لإنشاء كائنات (Objects). الكائنات هي نسخ من الكلاس وتحتوي على ال(Attributes)وال(Methods).

```
class Car
{
// الخصائص (Attributes)
public string brand;
public string model;
public int year;

// الدالة (Method)
public void DisplayInfo()
{
Console.WriteLine($"السيارة: {brand} {model}, سنة الصنع: {year}");
}
}
```

```
class Program
{
static void Main()
{
// إنشاء كائن جديد من الكلاس Car
Car myCar = new Car();

// تعيين القيم للخصائص
myCar.brand = "Toyota";
myCar.model = "Corolla";
myCar.year = 2022;

// استدعاء الدالة
myCar.DisplayInfo();
}
}
```

Constructor

ال(Constructor) هي دالة خاصة تُستدعى تلقائيًا عند إنشاء الكائن، وتُستخدم لتهيئة البيانات.

```
class Car{
public string brand;
public string model;
public int year;
// الدالة البانية (Constructor)
public Car(string carBrand, string carModel, int carYear){
brand = carBrand;
model = carModel;
year = carYear;}
public void DisplayInfo(){
Console.WriteLine($"السيارة: {brand} {model}, سنة الصنع: {year}"); }}
class Program
{
static void Main()
{
// إنشاء كائن باستخدام Constructor
Car myCar = new Car("Honda", "Civic", 2023);
myCar.DisplayInfo();
}
}
```

Properties

يمكن استخدام Properties بدلًا من public fields لجعل الكود أكثر أمانًا.

```
private string brand;

public string Brand {
get { return brand; }
set { brand = value; } }
```

Access Modifiers

(Access Modifiers):هي الكلمات المفتاحية التي تتحكم في إمكانية الوصول إلى المتغيرات (fields) أو الدوال (methods) أو الكلاسات (classes) داخل وخارج الكود.

1. private – خاص داخل الكلاس فقط
أكثر مستوى تقييدًا، يمكن الوصول إليه فقط داخل نفس الكلاس.
2. protected – متاح داخل الكلاس والميراث
يمكن الوصول إليه داخل نفس الكلاس وأيضًا في الكلاسات الموروثة (Child Classes).
3. internal – متاح داخل نفس الملف (Assembly)
يمكن الوصول إليه داخل نفس الملف البرمجي (Assembly)، لكنه غير متاح خارج هذا الملف.
4. protected internal – مزيج بين protected و internal
يمكن الوصول إليه داخل نفس المشروع (Assembly) وأيضًا في الكلاسات الموروثة خارج المشروع.
5. public – متاح في كل مكان
أقل مستوى من التقييد، يمكن الوصول إليه من أي مكان.

Access Modifiers

(Access Modifiers):هي الكلمات المفتاحية التي تتحكم في إمكانية الوصول إلى المتغيرات (fields) أو الدوال (methods) أو الكلاسات (classes) داخل وخارج الكود.

1. private – خاص داخل الكلاس فقط
أكثر مستوى تقييدًا، يمكن الوصول إليه فقط داخل نفس الكلاس.
2. protected – متاح داخل الكلاس والميراث
يمكن الوصول إليه داخل نفس الكلاس وأيضًا في الكلاسات الموروثة (Child Classes).
3. internal – متاح داخل نفس الملف (Assembly)
يمكن الوصول إليه داخل نفس الملف البرمجي (Assembly)، لكنه غير متاح خارج هذا الملف.
4. protected internal – مزيج بين protected و internal
يمكن الوصول إليه داخل نفس المشروع (Assembly) وأيضًا في الكلاسات الموروثة خارج المشروع.
5. public – متاح في كل مكان
أقل مستوى من التقييد، يمكن الوصول إليه من أي مكان.