**Mémo C#**

C# (C-Sharp) est un langage de programmation développé par Microsoft qui s'exécute sur le .NET Framework.

C # est utilisé pour développer des applications Web, des applications de bureau, des applications mobiles, des jeux et bien plus encore.

**Utilisation de la console**

Console.Write

Console.WriteLine

Console.ReadLine

Console.WriteLine("Enter your age:");

int age = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Your age is: " + age);

$ devant "…….." permet d’inclure {} dedans pour renvoyer la valeur de variables

string firstName = "John";

string lastName = "Doe";

string name = **$**"My full name is: **{**firstName**}** **{**lastName**}**";

Console.WriteLine(name); **→ My full name is: John Doe**

**Commentaires**

Commentaire 1 seule ligne : // (CTRL + K+ C dans visualstudio)

Commentaire sur plusieurs lignes : /\* \*/ (CTRL + Shift + / dans VisualStudio)

**Variables**

|  |  |  |
| --- | --- | --- |
| Data Type | Size | Description |
| int | 4 bytes | Stores whole numbers from -2,147,483,648 to 2,147,483,647 |
| long | 8 bytes | Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| float | 4 bytes | Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits |
| double | 8 bytes | Stores fractional numbers. Sufficient for storing 15 decimal digits |
| bool | 1 bit | Stores true or false values |
| char | 2 bytes | Stores a single character/letter, surrounded by single quotes |
| string | 2 bytes  per character | Stores a sequence of characters, surrounded by double quotes |

int myNum = 5;

int x = 5, y = 6, z = 50;

double myDoubleNum = 5.99D;

float myNum = 5.75F;

char myLetter = 'D';

bool myBool = true;

string myText = "Hello";

Exemple d’un Tableau d’entiers à 2 dimensions

int[,] numbers = { {1, 4, 2}, {3, 6, 8} };

Console.WriteLine(numbers[0, 2]); **→ 2**

**const** int myNum = 15; → constante

Les règles générales pour nommer les variables sont :

Les noms peuvent contenir des lettres, des chiffres et le caractère de soulignement (\_)

Les noms doivent commencer par une lettre

Les noms doivent commencer par une lettre minuscule et ne peuvent pas contenir d'espaces

Les noms sont sensibles à la casse ("myVar" et "myvar" sont des variables différentes)

# **Utilisation des opérateurs conditionnels et coalescents NULL en C#**

static void DisplayPerson(Person person){

Console.WriteLine(person**?**.FirstName);

Console.WriteLine(person**?**.LastName);

La partie à droite de ?n'est évaluée que si la partie à gauche n'est pas nulle. Sinon, le code renvoie null. Dans le cas ci-dessus, person?.FirstNameand person?.LastNameest évalué à null, mais il ne lève pas d'exception car il n'y a aucune tentative d'accès à un membre sur une référence null.

### Opérateur de fusion nulle (??)

Static void DisplayPerson(Person person){

Console.WriteLine(person?.FirstName **??** "Default First Name");

Console.WriteLine(person?.LastName **??** "Default Last Name");

}

**Conversion**

En C#, il existe deux types de casting :

Implicit Casting (automatiquement) - conversion d'un type plus petit en une taille de type plus grande

**char-> int-> long-> float->double**

Casting explicite (manuellement) - conversion d'un type plus grand en un type de taille plus petite

double-> float-> long-> int->char

Exemple de casting explicite :

double myDouble = 9.78;

int myInt = (int) myDouble; // Manual casting: double to int

Console.WriteLine(Convert.ToString(myInt)); // convert int to string

Console.WriteLine(Convert.ToDouble(myInt)); // convert int to double

Console.WriteLine(Convert.ToInt32(myDouble)); // convert double to int

Console.WriteLine(Convert.ToString(myBool)); // convert bool to string

**Les opérateurs arithmétiques :**

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Name | Description | Example |
| + | Addition | Adds together two values | x + y |
| - | Subtraction | Subtracts one value from another | x - y |
| \* | Multiplication | Multiplies two values | x \* y |
| / | Division | Divides one value by another | x / y |
| % | Modulus | Returns the division remainder | x % y |
| ++ | Increment | Increases the value of a variable by 1 | x++ |
| -- | Decrement | Decreases the value of a variable by 1 | x-- |

**les opérateurs d'affectation :**

|  |  |  |
| --- | --- | --- |
| Operator | Example | Same As |
| = | x = 5 | x = 5 |
| += | x += 3 | x = x + 3 |
| -= | x -= 3 | x = x - 3 |
| \*= | x \*= 3 | x = x \* 3 |
| /= | x /= 3 | x = x / 3 |
| %= | x %= 3 | x = x % 3 |
| &= | x &= 3 | x = x & 3 |
| |= | x |= 3 | x = x | 3 |
| ^= | x ^= 3 | x = x ^ 3 |
| >>= | x >>= 3 | x = x >> 3 |
| <<= | x <<= 3 | x = x << 3 |

**les opérateurs de comparaison :**

|  |  |  |
| --- | --- | --- |
| Operator | Name | Example |
| == | Equal to | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

**Les opérateurs logiques :**

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Name | Description | Example |
| && | Logical and | Returns True if both statements are true | x < 5 && x < 10 |
| || | Logical or | Returns True if one of the statements is true | x < 5 || x < 4 |
| ! | Logical not | Reverse the result, returns False if the result is true | !(x < 5 && x < 10) |

**Class Math**

|  |  |
| --- | --- |
| [**E**](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.e?view=net-7.0) | Représente la base logarithmique naturelle spécifiée par la constante e. |
| [**PI**](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.pi?view=net-7.0) | Représente le rapport de la circonférence d’un cercle à son diamètre, spécifié par la constante π. |
| [**Tau**](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.tau?view=net-7.0) | Représente le nombre de radians en un seul tour, spécifié par la constante, τ. |

|  |  |
| --- | --- |
| **[Abs()](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.abs?view=net-7.0" \l "system-math-abs(system-decimal))** | Retourne la valeur absolue d'un nombre |
| **[BigMul(Int32, Int32)](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.bigmul?view=net-7.0" \l "system-math-bigmul(system-int32-system-int32))** | Génère le produit intégral de deux nombres 32 bits. |
| **[Cbrt(Double)](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.cbrt?view=net-7.0" \l "system-math-cbrt(system-double))** | Retourne la racine cubique d’un nombre spécifié. |
| **[Ceiling()](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.ceiling?view=net-7.0" \l "system-math-ceiling(system-decimal))** | Retourne la plus petite valeur intégrale supérieure ou égale au nombre spécifié. |
| **[Clamp(](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.clamp?view=net-7.0" \l "system-math-clamp(system-byte-system-byte-system-byte))value,min,max)** | Retourne value, valeur limitée à la plage inclusive min-max. |
| **[Exp(Double)](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.exp?view=net-7.0" \l "system-math-exp(system-double))** | Retourne e élevé à la puissance spécifiée. |
| **[Floor(Decimal)](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.floor?view=net-7.0" \l "system-math-floor(system-decimal))** | Retourne la valeur du plus grand entier inférieur ou égal au nombre décimal spécifié. |
| **[Floor(Double)](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.floor?view=net-7.0" \l "system-math-floor(system-double))** | Retourne la valeur du plus grand entier inférieur ou égal au nombre à virgule flottante double précision spécifié. |
| **[FusedMultiplyAdd(Double, Double, Double)](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.fusedmultiplyadd?view=net-7.0" \l "system-math-fusedmultiplyadd(system-double-system-double-system-double))** | Retourne (x \* y) + z, arrondi en une seule opération ternaire. |
| **[IEEERemainder(Double, Double)](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.ieeeremainder?view=net-7.0" \l "system-math-ieeeremainder(system-double-system-double))** | Retourne le reste de la division d'un nombre spécifié par un autre. |
| **[Max(](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.max?view=net-7.0" \l "system-math-max(system-byte-system-byte))val1,val2)** | Retourne le plus grand de deux nombres |
| **[Min(](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.min?view=net-7.0" \l "system-math-min(system-byte-system-byte))val1,val2[)](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.min?view=net-7.0" \l "system-math-min(system-byte-system-byte))** | Retourne le plus petit de deux nombres |
| **[Pow(Double, Double)](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.pow?view=net-7.0" \l "system-math-pow(system-double-system-double))** | Retourne un nombre spécifié élevé à la puissance spécifiée. |
| **[Round(](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.round?view=net-7.0" \l "system-math-round(system-decimal))valeur)** | Arrondit une valeur décimale au nombre entier le plus proche |
| **[ScaleB(Double, Int32)](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.scaleb?view=net-7.0" \l "system-math-scaleb(system-double-system-int32))** | Retourne x \* 2 ^ n calculé efficacement. |
| **[Sign(](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.sign?view=net-7.0" \l "system-math-sign(system-decimal))valeur** | Retourne un entier qui indique le signe d’un nombre |
| **[Truncate(Decimal)](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.truncate?view=net-7.0" \l "system-math-truncate(system-decimal))** | Calcule la partie entière d'un nombre décimal spécifié. |
| **[Truncate(Double)](https://learn.microsoft.com/fr-fr/dotnet/api/system.math.truncate?view=net-7.0" \l "system-math-truncate(system-double))** | Calcule la partie entière d'un nombre à virgule flottante double précision spécifié. |

**Class String**

String.Length

String.ToUpper()

String.ToLower()

Extraire un carractère d’une chaine

string myString = "Hello";

Console.WriteLine(myString[1]); **→ e**

Extraire l’index d’une chaine dans une chaine

string myString = "Hello";

Console.WriteLine(myString.IndexOf("e")); **→ 1**

Extraire une chaine d’une chaine

string name = "John Doe";

int charPos = name.IndexOf("D");string lastName = name.Substring(charPos);Console.WriteLine(lastName); **→ Doe**

Le \ caractère d'échappement barre oblique inverse ( ) transforme les caractères spéciaux en caractères de chaîne :

|  |  |  |
| --- | --- | --- |
| Escape character | Result | Description |
| \' | ' | Single quote |
| \" | " | Double quote |
| \\ | \ | Backslash |

D'autres caractères d'échappement utiles en C# sont :

|  |  |
| --- | --- |
| Code | Result |
| \n | New Line |
| \t | Tab |
| \b | Backspace |

**Tests**

if (*condition1*)

{

*// block of code to be executed if condition1 is True*

}

else if (*condition2*)

{

*// block of code to be executed if condition2 is True*

}

else

{

*// block of code to be executed if condition2 is False*

}

**opérateur ternaire :** *variable = (condition)* ***?*** *expressionTrue : expressionFalse;*

switch(*expression*)

{

case x:

*// code block*

break;

case y:

*// code block*

break;

default:

*// code block*

break;

}

**Boucles**

**while** (*condition*)

{

*// code block to be executed*

}

**for** (int i = 0; i < 5; i++)

{

Console.WriteLine(i);

}

string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

**foreach** (string i in cars)

{

Console.WriteLine(i);

}

// Triun tableau de string

string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

Array.Sort(cars);

foreach (string i in cars)

{

Console.WriteLine(i);

}

**Méthodes / Fonctions**

static int MyMethod(int x, int y = 2) // Y a la valeur 2 par défaut

{

return x + y;

}

static void Main(string[] args)

{

int z = MyMethod(5, 3);

Console.WriteLine(z); **→ 8**

}

Exemple avec des paramètres nommés : au moment de l’appel, l’ordre n’a pas d’importance

static void MyMethod(string child1, string child2, string child3)

{

Console.WriteLine("The youngest child is: " + child3);

}

static void Main(string[] args)

{

MyMethod(child3: "John", child1: "Liam", child2: "Liam");

}

**surcharge** de méthode avec des paramètres différents ou de type différents

static int PlusMethod(int x, int y)

{

return x + y;

}

static double PlusMethod(double x, double y)

{

return x + y;

}

Exemple de définition d’une class **avec un constructeur**

class Car

{

public string model;

public string color;

public int year;

// Create a class constructor with multiple parameters

public Car(string modelName, string modelColor, int modelYear)

{

model = modelName;

color = modelColor;

year = modelYear;

}

static void Main(string[] args)

{

Car Ford = new Car("Mustang", "Red", 1969);

Console.WriteLine(Ford.color + " " + Ford.year + " " + Ford.model);

} }

**Modificateur d’accès**

|  |  |
| --- | --- |
| Modifier | Description |
| public | The code is accessible for all classes |
| private | The code is only accessible within the same class |
| protected | The code is accessible within the same class, or in a class that is inherited from that class. You will learn more about [inheritance](https://www.w3schools.com/cs/cs_inheritance.php) in a later chapter |
| internal | The code is only accessible within its own assembly, but not from another assembly. You will learn more about this in a later chapter |

**Encapsulation en utilisant les propriétés automatiques**

class Person

{

public string Name // property private avec getter / setter publique

**{ get; set; }**

}

class Program

{

static void Main(string[] args)

{

Person myObj = new Person();

myObj.Name = "Liam";

Console.WriteLine(myObj.Name);

}

}

En obligeant que toute instance créée de l’objet Person le Name soit défini

class Person

{

public required string Name // property private avec getter / setter publique

**{ get; set; }**

}

class Program

{

static void Main(string[] args)

{

Person myObj = new Person("Liam");

Console.WriteLine(myObj.Name);

}

}

**Héritage**

class Vehicle // base class (parent)

{

public string brand = "Ford"; // Vehicle field

public void honk() // Vehicle method

{

Console.WriteLine("Tuut, tuut!");

}

}

class Car : Vehicle // derived class (child)

{

public string modelName = "Mustang"; // Car field

}

class Program

{

static void Main(string[] args)

{

// Create a myCar object

Car myCar = new Car();

// Call the honk() method (From the Vehicle class) on the myCar object

myCar.honk();

// Display the value of the brand field (from the Vehicle class) and the value of the modelName from the Car class

Console.WriteLine(myCar.brand + " " + myCar.modelName);

}

}

**Polymorphisme**

class Animal // Base class (parent)

{

public virtual void animalSound()

{

Console.WriteLine("The animal makes a sound");

}

}

class Pig : Animal // Derived class (child)

{

public override void animalSound()

{

Console.WriteLine("The pig says: wee wee");

}

}

class Dog : Animal // Derived class (child)

{

public override void animalSound()

{

Console.WriteLine("The dog says: bow wow");

}

}

class Program

{

static void Main(string[] args)

{

Animal myAnimal = new Animal(); // Create a Animal object

Animal myPig = new Pig(); // Create a Pig object

Animal myDog = new Dog(); // Create a Dog object

myAnimal.animalSound(); **→ The animal makes a sound**

myPig.animalSound(); **→ The pig says: wee wee**

myDog.animalSound(); **→ The dog says: bow wow**

}

}

**Abstraction**

// Abstract class

abstract class Animal

{

// Abstract method (does not have a body)

public abstract void animalSound();

// Regular method

public void sleep()

{

Console.WriteLine("Zzz");

}

// Derived class (inherit from Animal)

class Pig : Animal

{

public override void animalSound()

{

// The body of animalSound() is provided here

Console.WriteLine("The pig says: wee wee");

}

}

class Program

{

static void Main(string[] args)

{

Pig myPig = new Pig(); // Create a Pig object

myPig.animalSound(); **→ The pig says: wee wee**

myPig.sleep(); // **→ Zzz**

}

}

**Interface**

Une interfaceest une " classe complètement abstraite ", qui ne peut contenir que des méthodes et des propriétés abstraites (avec des corps vides)

Il est considéré comme une bonne pratique de commencer par la lettre "I" au début d'une interface, car cela permet à vous-même et aux autres de vous souvenir plus facilement qu'il s'agit d'une interface et non d'une classe.

// Interface

interface IAnimal

{

void animalSound(); // interface method (does not have a body)

}

// Pig "implements" the IAnimal interface

class Pig : IAnimal

{

public void animalSound()

{

// The body of animalSound() is provided here

Console.WriteLine("The pig says: wee wee");

}

}

class Program

{

static void Main(string[] args)

{

Pig myPig = new Pig(); // Create a Pig object

myPig.animalSound(); **→ The pig says: wee wee**

}

}

**Enum**

Une enumest une "classe" spéciale qui représente un groupe de constantes (variables non modifiables/en lecture seule).

Enum est l'abréviation de "énumérations", ce qui signifie "spécifiquement répertorié".

class Program

{

enum Level

{

Low,

Medium,

High

}

static void Main(string[] args)

{

Level myVar = Level.Medium;

Console.WriteLine(myVar); **→ Medium**

}

}

## **Travailler avec des fichiers**

La Fileclasse de l' System.IOespace de noms, nous permet de travailler avec des fichiers :

La Fileclasse dispose de nombreuses méthodes utiles pour créer et obtenir des informations sur les fichiers. Par exemple:

|  |  |
| --- | --- |
| Method | Description |
| AppendText() | Appends text at the end of an existing file |
| Copy() | Copies a file |
| Create() | Creates or overwrites a file |
| Delete() | Deletes a file |
| Exists() | Tests whether the file exists |
| ReadAllText() | Reads the contents of a file |
| Replace() | Replaces the contents of a file with the contents of another file |
| WriteAllText() | Creates a new file and writes the contents to it. If the file already exists, it will be overwritten. |

using System.IO; // include the System.IO namespace

string writeText = "Hello World!"; // Create a text string

// Create a file and write the content of writeText to it

File.WriteAllText("filename.txt", writeText);

// Read the contents of the file

string readText = File.ReadAllText("filename.txt");

Console.WriteLine(readText); // Output the content

## **C# Exceptions**

Lors de l'exécution du code C#, différentes erreurs peuvent se produire : erreurs de codage faites par le programmeur, erreurs dues à une mauvaise saisie ou à d'autres choses imprévisibles.

Lorsqu'une erreur se produit, C# s'arrête normalement et génère un message d'erreur. Le terme technique pour cela est : C# lèvera une **exception** (lancera une erreur).

## **C# try and catch**

L' tryinstruction vous permet de définir un bloc de code à tester pour les erreurs pendant son exécution.

L' catchinstruction vous permet de définir un bloc de code à exécuter, si une erreur se produit dans le bloc try.

Try{

int[] myNumbers = {1, 2, 3};

Console.WriteLine(myNumbers[10]);

}catch (Exception e){

Console.WriteLine("Something went wrong.");

}finally{

Console.WriteLine("The 'try catch' is finished.");}

## The throw keyword

L' throwinstruction vous permet de créer une erreur personnalisée.

L' throwinstruction est utilisée avec une **classe d'exception** . Il existe de nombreuses classes d'exceptions disponibles en C# : ArithmeticException, FileNotFoundException, IndexOutOfRangeException, TimeOutException, etc :

static void checkAge(int age){

if (age < 18){

throw new ArithmeticException("Access denied - You must be at least 18

years old.");

}else{

Console.WriteLine("Access granted - You are old enough!");

}

}

static void Main(string[] args){

checkAge(15);

**→ System.ArithmeticException: 'Access denied - You must be at least 18 years old.'**

checkAge(20);

**→ Access granted - You are old enough!** }

**namespace**

Le mot clé namespace est utilisé pour déclarer une portée qui contient un ensemble d’objets connexes. Vous pouvez utiliser un espace de noms pour organiser les éléments de code et créer des types globaux uniques.

namespace SampleNamespace

{

class SampleClass { }

interface ISampleInterface { }

struct SampleStruct { }

enum SampleEnum { a, b }

delegate void SampleDelegate(int i);

namespace Nested

{

class SampleClass2 { }

}

}

L’exemple suivant montre comment appeler une méthode statique dans un espace de noms imbriqué.

namespace SomeNameSpace

{

public class MyClass

{

static void Main()

{

Nested.NestedNameSpaceClass.SayHello();

}

}

// a nested namespace

namespace Nested

{

public class NestedNameSpaceClass

{

public static void SayHello()

{

Console.WriteLine("Hello");

}

}

}

}

// Output: Hello

**Linq**

LINQ (Language-Integrated Query) est le nom d’un ensemble de technologies basé sur l’intégration de fonctions de requête directement dans le langage C#. Avec LINQ, une requête est une construction de langage de premier ordre, comme les classes, les méthodes et les événements. Vous pouvez écrire des requêtes pour des collections d’objets fortement typées à l’aide de mots clés de langage et d’opérateurs familiers. La famille de technologies LINQ fournit une expérience de requête cohérente pour les objets (LINQ to Objects), pour les bases de données relationnelles (LINQ to SQL) et pour le code XML (LINQ to XML).

Pour un développeur qui écrit des requêtes, la partie « intégrée au langage » la plus visible de LINQ est l’expression de requête. Les expressions de requête sont écrites selon une syntaxe de requête déclarative. En utilisant la syntaxe de requête, vous pouvez effectuer des opérations de filtrage, de classement et de regroupement sur des sources de données avec un minimum de code.

// Specify the data source.

int[] scores = { 97, 92, 81, 60 };

// Define the query expression.

IEnumerable<int> scoreQuery =

from score in scores

where score > 80

select score;

// Execute the query.

foreach (int i in scoreQuery)

{ Console.Write(i + " "); }

// Output: 97 92 81

**Pour travailler avec la date et l'heure en C#,**

créez un objet de la DateTimestructure à l'aide du nouveau mot-clé. Ce qui suit crée un DateTimeobjet avec la valeur par défaut.

Exemple : Créer un objet DateTime

// assigns default value 01/01/0001 00:00:00

DateTime dt = new DateTime();

Exemple : Définir la date et l'heure

//assigns default value 01/01/0001 00:00:00

DateTime dt1 = new DateTime();

//assigns year, month, day

DateTime dt2 = new DateTime(2015, 12, 31);

//assigns year, month, day, hour, min, seconds

DateTime dt3 = new DateTime(2015, 12, 31, 5, 10, 20);

//assigns year, month, day, hour, min, seconds, UTC timezone

DateTime dt4 = new DateTime(2015, 12, 31, 5, 10, 20, DateTimeKind.Utc);

La DateTimestructure comprend des champs statiques, des propriétés et des méthodes. L'exemple suivant illustre des champs et des propriétés statiques importants.

Exemple : Champs statiques

DateTime currentDateTime = DateTime.Now; //returns current date and time

DateTime todaysDate = DateTime.Today; // returns today's date

DateTime currentDateTimeUTC = DateTime.UtcNow;// returns current UTC date and time

DateTime maxDateTimeValue = DateTime.MaxValue; // returns max value of DateTime

DateTime minDateTimeValue = DateTime.MinValue; // returns min value of DateTime

## **Période**

TimeSpan est une structure utilisée pour représenter le temps en jours, heures, minutes, secondes et millisecondes.

Exemple : Période

DateTime dt = new DateTime(2015, 12, 31);

TimeSpan ts = new TimeSpan(25,20,55);

DateTime newDate = dt.Add(ts)

Console.WriteLine(newDate);//1/1/2016 1:20:55 AM

## Convertir une chaîne en DateTime

Une chaîne de date et d'heure valide peut être convertie en DateTimeobjet àl'aidedes méthodes [Parse()](https://docs.microsoft.com/en-us/dotnet/api/system.datetime.parse?view=netframework-4.8), [ParseExact()](https://docs.microsoft.com/en-us/dotnet/api/system.datetime.parseexact?view=netframework-4.8), [TryParse()](https://docs.microsoft.com/en-us/dotnet/api/system.datetime.tryparse?view=netframework-4.8) et [TryParseExact()](https://docs.microsoft.com/en-us/dotnet/api/system.datetime.tryparseexact?view=netframework-4.8) .

Exemple:

var str = "5/12/2020";

DateTime dt;

var isValidDate = DateTime.TryParse(str, out dt);

if(isValidDate)

Console.WriteLine(dt);

else

Console.WriteLine($"{str} is not a valid date string");

**Gestion des dépendances**

Le système de gestion des dépendances pour C# s’appelle Nuget.

Exemple installation de la dépendance Newtonsoft.Json

Sélectionnez **Projet**>**Gérer les packages NuGet**.

1. Dans la page **Gestionnaire de package NuGet** , choisissez **nuget.org** comme **source de package**.
2. Dans l’onglet **Parcourir** , recherchez Newtonsoft.Json, sélectionnez **Newtonsoft.Json** dans la liste, puis sélectionnez **Installer**.

**Désérialiser Json en un objet**

public class Account

{

public string Email { get; set; }

public bool Active { get; set; }

public DateTime CreatedDate { get; set; }

public IList<string> Roles { get; set; }

}

string json = @"{

'Email': 'james@example.com',

'Active': true,

'CreatedDate': '2013-01-20T00:00:00Z',

'Roles': [

'User',

'Admin'

]

}";

Account account = JsonConvert.DeserializeObject<Account>(json);

Console.WriteLine(account.Email);

// james@example.com