NOVEI FORMATION

LA MANU

L'ÉCOLE DES MÉTIERS DU NUMÉRIQUE
Manufacture de compétences

SUPPORT APPRENANT

Application console

CONFIDENTIEL

Application console

Ce que l'on a vu…

Le C# est un langage de programmation populaire, développé par la société américaine Microsoft au début des années 2000. L'utilisation de ce langage seul est assez limitée, c'est pourquoi il est souvent couplé au framework .NET ce qui offre plus de possibilités. Le framework .NET est créé en même temps que le langage C# par Microsoft.

Le C# est un langage dérivé du langage C++, et il est aussi très proche du langage Java pour certains concepts et au niveau de sa syntaxe. Le C++ est un langage compilé, c'est-à-dire que le code écrit par le développeur va être compilé dans un langage machine (binaire) exécutable par l'ordinateur. Le Java, à l'inverse, est un langage interprété, c'est-à-dire que le code source est transformé par un interpréteur. Le code est exécuté instruction par instruction et le résultat obtenu est immédiatement renvoyé par le programme interpréteur au navigateur, ce qui peut ralentir l'exécution par rapport à un langage compilé. Le C# quant à lui, est un **c**ode en **l**angage **i**ntermédiaire (CIL) entre le langage compilé et le langage interprété. Le code est compilé non pas en langage machine mais en langage intermédiaire qui va être exécuté et compilé en langage machine par le CLR (**C**ommon **L**anguage **R**untime), ce qui permet d'améliorer la rapidité d'exécution par rapport au langage interprété.

Le développement en C#.NET se fait généralement sur l'IDE (**I**ntegrated **D**evelopment **E**nvironment ou environnement de développement intégré) Visual Studio qui est l'IDE de Microsoft. Plusieurs types d'applications peuvent être créées en C#.NET sur Visual Studio : application console, application web, application WPF…

L'application console est la plus simple pour commencer l'apprentissage du C#.NET. Elle permet de créer des applications simples, avec très peu de design, qui s'exécutent sur l'écran noir de la console.

A la création d'une application console sur Visual Studio, on retrouve plusieurs éléments :

- Une solution : *nomDeLaSolution*.sln, c'est un répertoire de projets. Une solution peut contenir un ou plusieurs projets.
- Un projet : *nomDuProjet*.csproj
- Un fichier source C# : Program.cs, le fichier qui contient le code.
- D'autres éléments : Propriétés, Références, App.config.

Le fichier Program.cs contient du code généré par Visual Studio à sa création. Il est composé de la structure suivante :

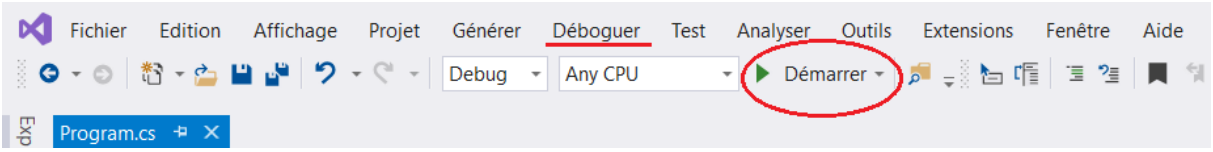
```
//ensemble de références qui peuvent être utilisées dans l'application
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

//espace de nom du nom du projet créé
namespace NomProjet
{
  //classe Program
  class Program
```

```
  {
    //méthode static Main qui ne retourne rien (void)
    static void Main(string[] args)
    {
      //votre code
    }
  }
}
```

La méthode **Main()** est très importante dans une application console car il s'agit du point d'entrée de l'application, c'est-à-dire que c'est à partir de cette méthode que les instructions sont exécutées. Le mot-clé *static* précédant la méthode permet d'indiquer qu'elle sera accessible de partout, sans cela l'application ne peut pas s'exécuter.

Lorsque l'on souhaite exécuter son code pour l'afficher dans la console, on peut utiliser le bouton « Démarrer », ou dans l'onglet Déboguer « Démarrer le débogage » (F5) ou « Exécuter sans débogage » (ctrl + F5). Cependant quand on clique sur « Démarrer » ou « Démarrer le débogage », la console s'ouvre, exécute le code et se referme aussitôt. Pour éviter cela, on peut « Exécuter sans débogage » ou simplement placer la commande **Console.ReadKey()** comme dernière instruction de la méthode *Main()*. Elle permet de maintenir la console ouverte jusqu'à ce que l'utilisateur tape sur une touche du clavier quelle qu'elle soit.



Variables

Le C# est un langage typé, c'est-à-dire qu'il faut préciser le type des éléments que l'on déclare. Ainsi lorsqu'on déclare une variable il faut indiquer le type de la valeur qui lui sera attribuée.

Type	Description
string	Chaîne de caractères.
int	Integer, nombre entier de -2147483648 à 2147483647.
float	Nombre décimal de $\pm 1.5 \times 10^{-45}$ à $\pm 3.4 \times 10^{38}$, précision 6-9 chiffres significatifs.
double	Nombre décimal de $\pm 5.0 \times 10^{-324}$ à $\pm 1.7 \times 10^{308}$, précision 15-17 chiffres significatifs.
decimal	Nombre décimal de -7.9×10^{28} à 7.9×10^{28}, précision 28-29 chiffres significatifs.
bool	Boolean, booléen (vrai ou faux).

Il est possible de déclarer une variable et de lui attribuer directement une valeur, ou bien déclarer une variable puis lui assigner une valeur plus loin dans le code. Le type de la variable ne se précise qu'à la déclaration de celle-ci. Lorsqu'on manipule cette variable par la suite il n'est pas nécessaire de la précéder de son type.

Le nom d'une variable commence toujours par une lettre en minuscule, et il ne peut pas contenir de caractères spéciaux ni d'espace. Pour une meilleure compréhension on utilise des noms explicitent et

le camelCase lorsqu'un nom de variable est composé de plusieurs mots. De plus, le nom d'une variable ne peut pas être le même que celui d'un type. Ainsi, il n'est pas possible de nommer une variable string ou int par exemple.

Une variable est donc définie par un type, un nom et elle contient une valeur. Sa syntaxe est la suivante :

```
type nomVariable = valeur;
```

Pour afficher un élément dans la console, il faut utiliser la commande **Console.WriteLine(*paramètre*)**. L'élément que l'on souhaite afficher se place en paramètre de la méthode *WriteLine()*. Il peut s'agir d'une chaîne de caractères, d'une variable ou même de la concaténation d'une chaîne de caractères et de variables. Différentes syntaxes peuvent être employées.

Exemple :

```
static void Main(string[] args)
{
   string firstname = "Jean";
   string lastname = "Dupont";
   Console.WriteLine("Bonjour ! ");
   Console.WriteLine("Bienvenue " + firstname + " " + lastname);
   Console.WriteLine($"Bienvenue {firstname} {lastname}");
   Console.WriteLine("Bienvenue {0} {1}", firstname, lastname);
   Console.ReadKey();
}
```

Pour récupérer la valeur saisie par un utilisateur dans la console, on utilise la commande **Console.ReadLine()**. Elle permet de lire la ligne saisie par l'utilisateur et d'enregistrer cette valeur dans une variable.

Exemple :

```
static void Main(string[] args)
{
   Console.WriteLine("Veuillez saisir votre prénom : ");
   string firstname = Console.ReadLine();
}
```

La valeur retournée par la commande *Console.ReadLine()* est toujours de type *string*. Si le résultat attendu est un nombre il faudra le convertir avec la méthode ***type*.Parse()** par exemple. Il est cependant conseillé d'utiliser la méthode ***type*.TryParse()** qui retourne un booléen et le résultat de la conversion selon si cette dernière est possible ou non. Ainsi, cela évite de lever une exception bloquant l'application et permet d'afficher un message d'erreur à l'utilisateur.

Pour convertir un nombre ou une date en chaîne de caractères par exemple, il est possible d'utiliser la méthode **ToString()**.

Il existe plusieurs méthodes prédéfinies par le framework .NET et qui peuvent être utilisées selon les besoins, notamment pour manipuler des dates, faire des calculs, ajouter du style…

Quelques méthodes et commandes…

Méthodes/Commandes	Description
Console.ReadKey();	Obtient la touche appuyée par l'utilisateur dans la console.
Console.WriteLine();	Affiche un élément dans la console.

Console.ReadLine();	Obtient la ligne saisie par l'utilisateur dans la console.
Console.Clear();	Efface l'affichage présent dans la console.
Console.SetCursorPosition(*x*,*y*);	Définit la position du curseur (et donc du texte).
Console.ForegroundColor = ConsoleColor.*couleur*;	Permet de définir une couleur de texte.
Console.ResetColor();	Redéfinit les couleurs par défaut (texte et fond).
Console.BackgroundColor = ConsoleColor.*couleur*;	Permet de définir une couleur de fond.
type.Parse(*variable*);	Convertit la variable dans le type défini.
type.TryParse(*variableAConvertir*, out *type variableConvertie*);	Convertit une valeur dans un type donné et stocke la conversion dans une variable, retourne un booléen indiquant si la conversion a réussi ou échoué.
variable.ToString();	Convertit la variable en chaîne de caractères.
DateTime.Now.ToLongDateString();	Affiche la date du jour au format *lundi 22 juillet 2019*.
DateTime.Now.ToShortDateString();	Affiche la date du jour au format *22/07/2019*.
DateTime.Now.Year;	Année en cours.
Environment.NewLine ou \n	Retour à la ligne.
\t	Tabulation.
Math.Round()	Arrondit un nombre.
Math.Sqrt()	Obtient la racine carrée d'un nombre.
Math.Pow()	Obtient la puissance d'un nombre.
Math.PI	Valeur de pi (π).
variable.ToLower()	Convertit la variable en minuscules.
variable.ToUpper()	Convertit la variable en majuscules.
nomTableau.Length	Obtient la longueur du tableau (le nombre d'éléments qu'il contient).
nomTableau.Take(*n*)	Prend *n* éléments du tableau.
new Random().Next(*valeurMinimaleInclue*, *valeurMaximaleExclue*);	Génère un nombre aléatoire compris entre 2 valeurs.

Conditions

Les conditions dans le langage C# fonctionnent de la même manière que pour la majorité des langages. La syntaxe est la suivante :

```
if (condition)
{
   instructions
}
else if (condition)
{
   instructions
}
else
{
   instructions
}
```

Il est tout à fait possible de faire des conditions multiples en posant plusieurs conditions à la fois dans un *if* ou un *else if*, en séparant les conditions avec les opérateurs de comparaison logique && ou ||.

Opérateur de comparaison	Description
==	Egalité
===	Strictement égal (en valeur et type)
!=	Différence
>	Strictement supérieur
>=	Supérieur ou égal
<	Strictement inférieur
<=	Inférieur ou égal
&&	ET (opérateur de comparaison logique)
||	OU (opérateur de comparaison logique)
!	Négation

Dans certains cas, il peut être plus judicieux d'utiliser un *switch* plutôt que de multiplier les *if* et *else if*. En effet, lorsque les conditions correspondent à des cas précis où une variable pourrait être égale à une valeur parmi un choix de plusieurs valeurs particulières par exemple, on optera pour un *switch*. Généralement on peut utiliser un *switch* à partir de 3 conditions à traiter. Dans chaque *case*, il faut veiller à terminer par une instruction *break* afin d'éviter d'exécuter les instructions *case* suivantes. Il faut également terminer par un *default* qui sera exécuté si tous les autres cas ont échoué.

Tableaux, Listes, Enumérations

Tableaux

Les tableaux sont un type qui se caractérise par des crochets. Et comme pour les variables, il faut indiquer de quel type sont les données contenues par le tableau. Ainsi, pour un tableau contenant des chaînes de caractères on utilisera **string[]**, et pour un tableau de nombres entiers on utilisera **int[]**. Il est possible d'attribuer directement des valeurs à la déclaration d'un tableau :

```
type[] nomTableau = new type[] { valeur1, valeur2, valeur3 } ;
```

Un index, indice ou encore clé, est associé par défaut à chaque valeur du tableau. Cet index est un nombre entier. La 1ère valeur du tableau a toujours l'index 0, la 2ème valeur a l'index 1, la 3ème valeur l'index 2, et ainsi de suite. Il est également possible de déclarer un tableau en précisant le nombre de données qu'il contiendra, puis lui assigner des valeurs par la suite :

```
type[] nomTableau = new type[nombreDeValeur];
nomTableau[index] = valeur1 ;
nomTableau[index] = valeur2 ;
nomTableau[index] = valeur3 ;
```

Les tableaux à une dimension sont les plus fréquents mais il est parfois possible d'avoir besoin d'un tableau à plusieurs dimensions. Un tableau à 2 dimensions se déclare de la manière suivante :

```
type[,] nomTableau = new type[nombreDeLignes, nombreDeColonne]
{
   { valeurLigne1Colonne1, valeurLigne1Colonne2 },
   { valeurLigne2Colonne1, valeurLigne2Colonne2 },
   { valeurLigne3Colonne1, valeurLigne3Colonne2 },

} ;
```

Listes

A la différence des tableaux qui ont une taille fixée à leur déclaration, les listes n'ont pas cette limite. Il est possible d'ajouter ou de supprimer autant d'éléments qu'on le souhaite dans une liste. En effet,

pour utiliser une liste on peut la déclarer sans indiquer sa taille, puis lui ajouter des valeurs avec la méthode **Add()** :

```
List<type> nomListe = new List<type>();
nomListe.Add(valeur1);
nomListe.Add(valeur2);
nomListe.Add(valeur3);
//ou déclaration et initialisation d'une liste
List<type> nomListe = new List<type> { valeur1, valeur2, valeur3 };
```

Comme pour les tableaux, les valeurs d'une liste sont définies par un index qui commence à 0. Avec la méthode *Add()* les valeurs sont ajoutées automatiquement aux indices dans l'ordre croissant. Pour ajouter une valeur à un index en particulier, il faut utiliser la méthode ***nomTableau*.Insert(*index*, *valeur*);**. Il est aussi possible d'utiliser la méthode ***nomTableau*.IndexOf(*valeur*);** pour connaitre l'index d'une valeur dans la liste. Cela peut être pratique pour savoir l'indice d'une valeur qu'on souhaiterait afficher dans la console avec la commande **Console.WriteLine(*nomListe*[*index*]);** par exemple.

Enumérations

Les énumérations sont un type particulier, elles se déclarent à l'extérieur de la méthode *Main()*. Elles contiennent une liste de noms symboliques dont toutes les valeurs sont fixes. Les valeurs d'une énumération sont des entiers allant de 1 en 1 et commençant par défaut à 0. Cependant il est possible de forcer toutes ou certaines valeurs de l'énumération. Si une valeur n'est pas forcée elle prend la valeur précédente +1. Par convention le nom d'une énumération commence par une majuscule et ne contient pas d'espaces.

Exemple :

```
enum NomEnumération
{
   Valeur1, //Valeur1 vaut 0
   Valeur2, //Valeur2 vaut 1
   Valeur3 = 5, //Valeur3 vaut 5
   Valeur4, //Valeur4 vaut 6
   Valeur5, //Valeur5 vaut 7
   Valeur6 = 10, //Valeur6 vaut 10
   Valeur7, //Valeur7 vaut 11
}
```

Il est ensuite possible d'accéder à un élément de l'énumération grâce à sa valeur (différentes syntaxes sont possibles) ou en créant une variable du type de l'énumération.

```
NomEnumération nomVariable = NomEnumération.Valeur3 ;
Console.WriteLine(nomVariable);

Console.WriteLine(Enum.GetName(typeof(NomEnumération), 5));
Console.WriteLine(Enum.GetNames(typeof(NomEnumération)).GetValue(5));
Console.WriteLine((NomEnumération)5);
```

Boucles

Les boucles servent à répéter une instruction un certain nombre de fois ou tant qu'une condition n'est pas atteinte par exemple. Il existe différents types de boucles.

Boucle while

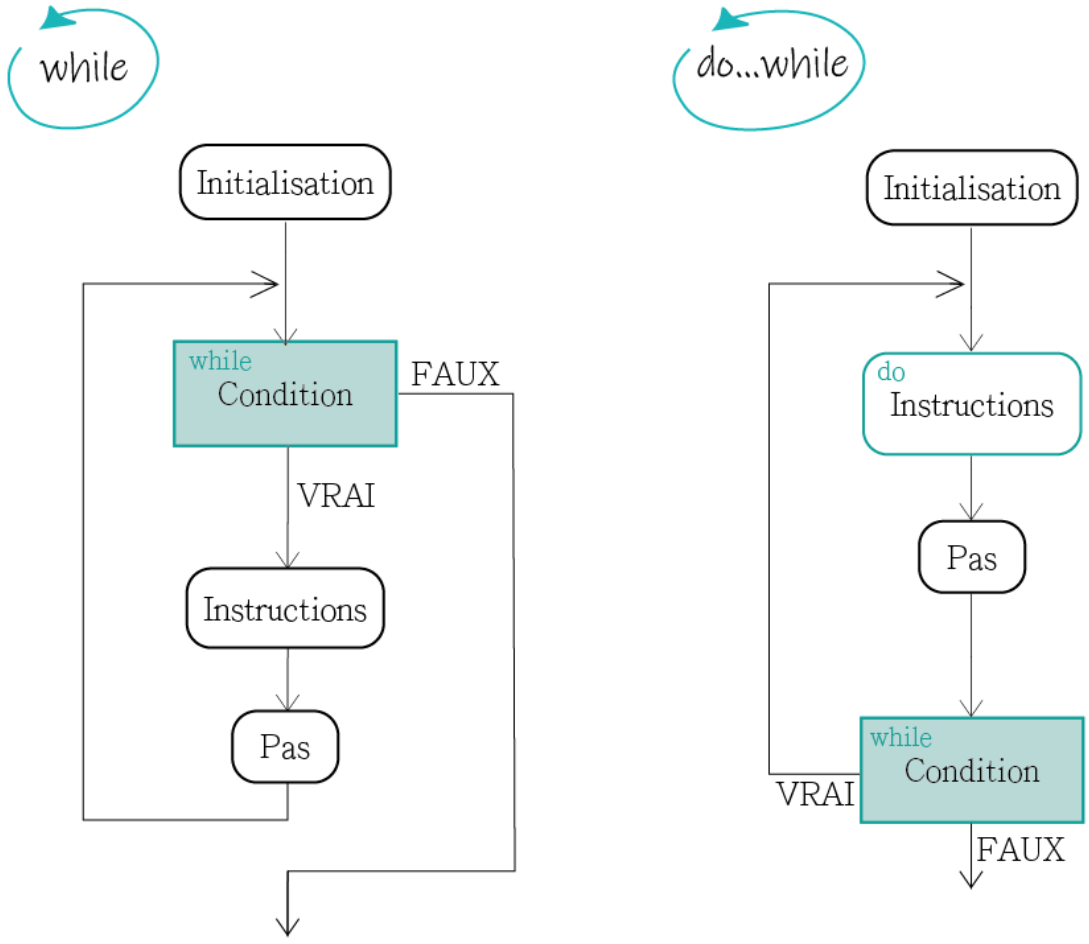
La boucle **while** peut être traduit en français par « tant que ». Sa syntaxe est la suivante :

```
initialisation ;
while(condition)
{
   instructions
   pas
}
```

Il est possible de trouve une autre syntaxe pour la boucle while :

```
initialisation ;
do
{
   instructions
   pas
}
while(condition) ;
```

Avec ce sens d'écriture les instructions sont exécutées au moins une fois même si la condition est déjà atteinte car celle-ci n'est vérifiée qu'après les instructions, contrairement à la syntaxe classique qui vérifie d'abord la condition avant d'exécuter les instructions.



La boucle *while* est simple et flexible, elle s'utilise plutôt lorsque le nombre d'itération de la boucle est inconnu. Cependant leur syntaxe fait qu'il y a plus de risque d'oublier un élément et ainsi faire une boucle infinie, surtout lorsqu'on manipule des valeurs numériques.

Boucle for

La boucle **for** peut être traduit en français par « pour ». Elle s'utilise principalement lorsque l'on connait le nombre de fois que l'on souhaite réaliser une instruction. Sa syntaxe est la suivante :

```
for(initialisation ; condition ; pas)
{
   instructions
}
```

L'avantage d'une boucle *for* est que l'on retrouve toutes les informations importantes, à savoir la valeur initiale, la condition et le pas, au même niveau. Ce qui n'est pas le cas avec une boucle *while* avec laquelle ces informations sont plus dispersées. Il y a donc plus de risques d'oublier un élément (initialisation ou pas) et de faire une boucle infinie. Autant que possible on préfèrera privilégier les boucles *for* aux boucles *while* pour limiter les risques de boucles infinies.

Boucle foreach

La boucle **foreach** peut être traduit en français par « pour chaque élément dans ce tableau ». Il s'agit de la boucle la plus adaptée pour parcourir un tableau. Sa syntaxe est la suivante :

```
foreach(type élément in nomTableau)
{
   instructions (par ex : Console.WriteLine(élément);)
}
```

Le nom de la variable *élément* précédant le « in » est un nom que l'on donne de façon arbitraire et qui contiendra consécutivement la valeur de chaque élément du tableau parcouru. La portée de cette variable se limite à la boucle *foreach*.

Liens utiles

https://docs.microsoft.com/fr-fr/dotnet/csharp/programming-guide/
https://www.supinfo.com/articles/single/6382-qu-est-ce-que-c
http://sdz.tdct.org/sdz/apprenez-a-developper-en-c.html
https://perso.esiee.fr/~perretb/I3FM/POO1/index.html
https://docs.microsoft.com/fr-fr/dotnet/standard/base-types/standard-date-and-time-format-strings
https://docs.microsoft.com/fr-fr/dotnet/api/system.math.round?view=netframework-4.7.2#System_Math_Round_System_Double_System_Int32_
https://docs.microsoft.com/fr-fr/dotnet/api/system.math.sqrt?view=netframework-4.7.2
https://docs.microsoft.com/fr-fr/dotnet/api/system.math.pow?view=netframework-4.7.2
https://docs.microsoft.com/fr-fr/dotnet/csharp/language-reference/keywords/switch
https://openclassrooms.com/en/courses/1526901-apprenez-a-developper-en-c/
https://openclassrooms.com/fr/courses/218202-apprenez-a-programmer-en-c-sur-net
https://openclassrooms.com/fr/courses/392266-developpement-c-net/391031-le-fonctionnement-de-net
https://docs.microsoft.com/fr-fr/dotnet/csharp/programming-guide/enumeration-types
https://docs.microsoft.com/fr-fr/dotnet/api/system.enum?view=netframework-4.7.2
https://docs.microsoft.com/fr-fr/dotnet/csharp/tour-of-csharp/enums

Bonnes pratiques

Rappel des principales bonnes pratiques	
☐	Indiquer le type des variables à la déclaration.
☐	Une instruction switch doit toujours contenir une étiquette default, et une instruction break dans chaque étiquette case.
☐	Prendre en compte tous les cas possibles pour les conditions.
☐	Utiliser la méthode TryParse() pour vérifier la validité des valeurs numériques saisies par l'utilisateur.
☐	Les énumérations se déclarent en dehors de la méthode Main().
☐	Les énumérations se nomment avec une majuscule pour la première lettre.
☐	Privilégier l'utilisation de boucle foreach pour parcourir des tableaux.
☐	Privilégier les boucles for plutôt que les boucles while autant que possible car il y a moins de risque d'oublier un élément.
☐	Nommage des fichiers, variables… sans accent, sans caractère spécial (underscore, espace…). Utilisation du camelCase.
☐	Vocabulaire spécifique : • Variable • Console • Solution • Projet • Auto-complétions • Méthode • Opérateurs de comparaison • Boucles while / do…while / for / foreach • Incrémentation / Décrémentation