Module 2

Utiliser les instructions C#

Sommaire

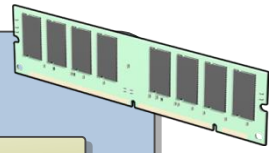
- Déclarer des Variables et Assigner des Valeurs
- Utiliser les Expressions les Opérateurs
- Créer et Utiliser des Arrays
- Utiliser des Instructions de Décision
- Utiliser des Instructions d’Itération

Leçon 1: Déclarer des Variables et Assigner des Valeurs

- Qu’est-ce qu’une Variable?
- Quels sont les types de données?
- Déclarer et Assigner
- Qu’est-ce que le Scope d’une Variable?
- Convertir une Value dans un Type de Données Différent
- Variables en Lecture-Seule et Constantes

Qu’est-ce qu’une Variable?

Les variables stockent les valeurs requisent par l’ application dans un emplacement mémoire temporaire



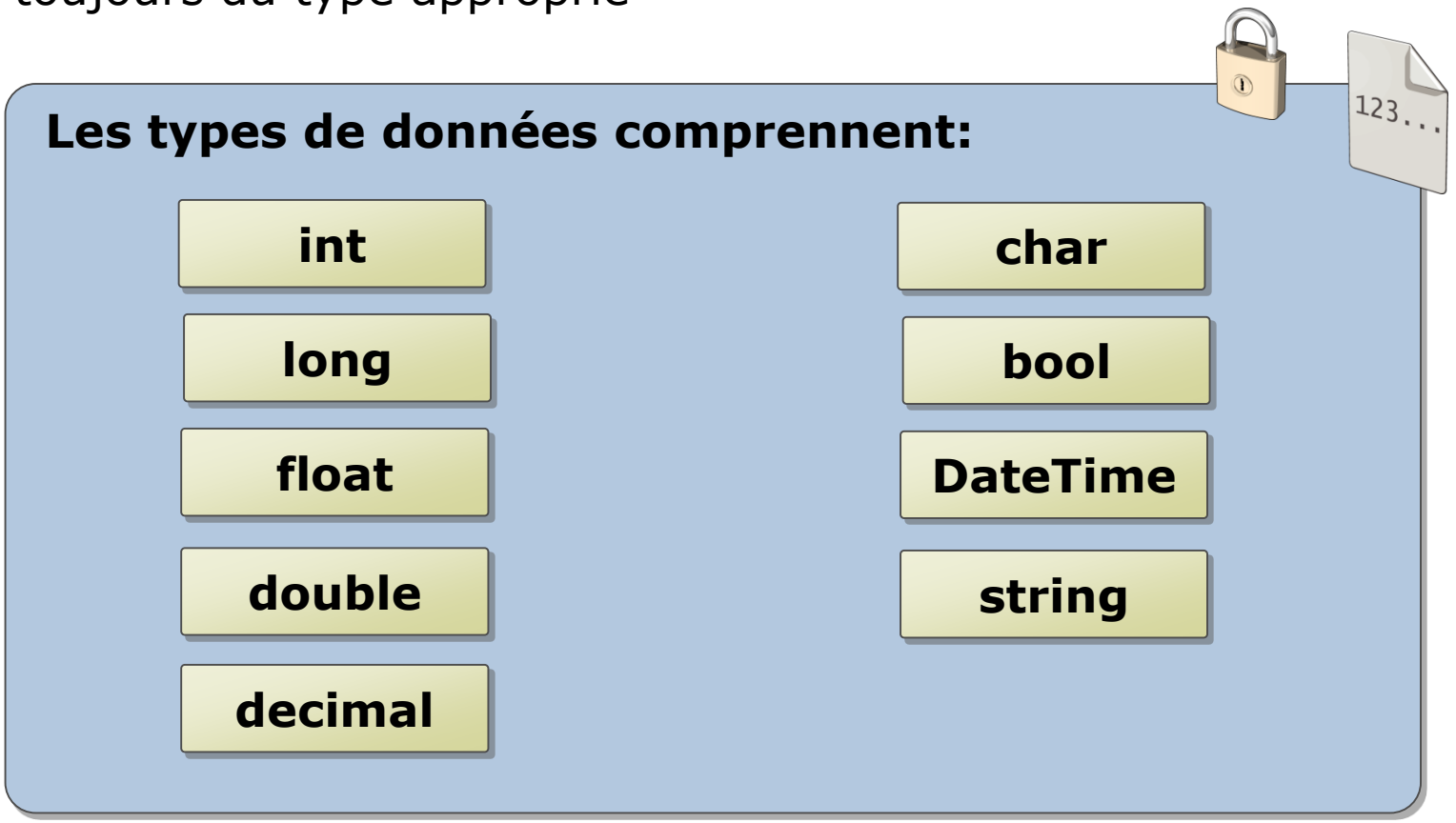
Les variables possèdent les caractéristiques suivantes:

- Nom
- Adresse
- Type de données
- Valeur
- Scope
- Durée de vie

Quels sont les types de données?

C# un langage type-safe

Le compilateur garantit que les valeurs stockées dans des variables sont toujours du type approprié



Déclarer et Assigner

Avant d'utiliser une variable, vous dever la déclarer

```
DataType variableName;
…
DataType variableName1, variableName2;
…
DataType variableName = new DataType();
```

Une fois la variable déclarée, vous pouvez lui assigner une valeur

```
variableName = Value;
…
DataType variableName = Value;
```

NOTE: Le nom de la Variable est reconnu comme un identificateur. Les identificateurs doivent:

- Contenir uniquement des lettre, digits, et underscore
- Commencer par une lettre ou un underscore
- Ne pas être un mot clé réserver à l'utilisation de C#

Qu'est-ce que le Scope d'une Variable?

Scope de Block

```
if (length > 10)
{
    int area = length * length;
}
```

Scope de Procedure

```
void ShowName()
{
    string name = "Bob";
}
```

Scope de Classe

```
private string message;

void SetString()
{
    message = "Hello World!";
}
```

Scope de niveau Namespace

```
public class CreateMessage
{
  public string message
    = "Hello";
}

public class DisplayMessage
{
   public void ShowMessage()
   {
       CreateMessage newMessage
         = new CreateMessage();
       MessageBox.Show(
         newMessage.message);

   }
}
```

Convertir une Value dans un Type de Données Différent

Convertion Implicite

Réalisée Automatiquement par le common language runtime

```
int a = 4;
long b;
b = a;          // Implicit conversion of int to long
```

Convertion Explicite

Nécessite l'écriture de code pour réaliser la convertion

```
DataType variableName1 = (castDataType) variableName2;
…
int count = Convert.ToInt32("1234");
...
int number = 0;
if (int.TryParse("1234", out number)) {// Conversion succeeded }
```

Variables en Lecture-Seule et Constantes

Variable en lecture-seule

Déclarée avec le mot clé **readonly**
Initialisée au "run time"

```
readonly string currentDateTime = DateTime.Now.ToString();
```

Constante

Declarée avec le mot clé **const**
Initialisée à la compilation

```
const double PI = 3.14159;
int radius = 5;
double area = PI * radius * radius;
double circumference = 2 * PI * radius;
```

Leçon 2: Utiliser les Expressions les Opérateurs

- Qu’est-ce qu’une Expression?
- Qu’est-ce que les Opérateurs?
- Spécifier la priorité des opérateurs
- Bonnes pratiques pour l’exécution de concaténation de chaînes

Qu’est-ce qu’une Expression?

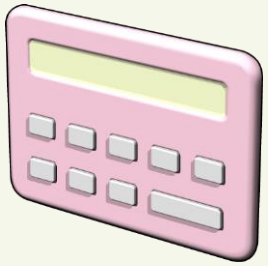
Les expressions sont les constructions fondamentales qui vous permettent d’évaluer et de manipuler des données

```
a + 1

(a + b) / 2

"Answer: " + c.ToString()

b * System.Math.Tan(theta)
```



Qu’est-ce que les Opérateurs?

Les opérateurs sont des séquences de caractères que vous utilisez pour définir des opérations qui doivent être exécutées sur les opérandes

- Arithmetique **+**, **-**, *****, **/**, **%**
- Assignation **=**, **+=**, **-=**, ***=**, **/=**
- Incrément, décrément **++**, **--**
- Bit shift **<<**, **>>**
- Comparaison **==**, **!=**, **<**, **>**, **<=**, **>=**, **is**
- Information de Type **sizeof**, **typeof**
- Concaténation de **+**
- Concaténation de Déléguer **+**, **-**
- Logique/bitwise **&**, **|**, **^**, **!**, **~**, **&&**, **||**
- Overflow **checked**, **unchecked**
- Index **[ ]**
- Indirection, Adresse *****, **->**, **[ ]**, **&**
- Casting **( )**, **as**
- Condition (opérateur ternaire) **?:**

Spécifier la priorité des opérateurs

Plus élevé

Moins élevé

++, -- (prefixes), +, - (unary), !, ~
***, /, %**
+, -
<<, >>
<, >, <=, >=
==, !=
&
^
|
&&
||
Assignment operators
++, -- (suffixes)

Bonnes pratiques pour l'exécution de concaténation de chaînes

La concaténation de plusieurs chaînes dans c# est simple à réaliser en utilisant l'opérateur +

```
string address = "23";
address = address + ", Oxford Street";
address = address + ", Thornbury";
```

Ceci est considéré comme une mauvaise pratique car les chaînes sont immuables

s

```
StringBuilder address = new StringBuilder();

address.Append("23");
address.Append(", Oxford Street");
address.Append(", Thornbury");

string concatenatedAddress = address.ToString();
```

Leçon 3: Créer et Utiliser des Arrays

- Qu’est-ce qu’un Array?
- Création et Initialisation d’Arrays
- Propriétés et Méthodes Exposées par les Arrays
- Accéder aux données d’un Array

Qu’est-ce qu’un Array?

Un Array est une séquence d’éléments qui sont regroupés

Un Array comprend les fonctionnalités suivantes:

- Chaque élément du tableau contient une valeur
- Les arrays sont indexés en Base zéro
- La longueur d’un tableau est le nombre total d’éléments qu’il peut contenir
- La limite inférieure d’un tableau est l’index du premier élément
- Les tableaux peuvent être unidimensionnels, multidimensionnels ou en escalier
- Le rang d’un tableau est le nombre de dimensions dans le tableau

Création et Initialisation d'Arrays

Un tableau peut avoir plusieurs dimensions

Unique

```
Type[] arrayName = new Type[ Size ];
```

Multiple

```
Type[ , ] arrayName = new Type[ Size1, Size2];
```

En escalier

```
Type [][] JaggedArray = new Type[size][];
```

Propriétés et Méthodes Exposées par les Arrays

Length

```
int[] oldNumbers = { 1, 2, 3, 4, 5 };
int numberCount = oldNumbers.Length;
```

Rank

```
int[] oldNumbers = { 1, 2, 3, 4, 5 };
int rank = oldNumbers.Rank;
```

CopyTo()

```
int[] oldNumbers = { 1, 2, 3, 4, 5 };
int[] newNumbers = new
    int[oldNumbers.Length];
oldNumbers.CopyTo(newNumbers, 0);
```

Sort()

```
int[] oldNumbers = { 5, 2, 1, 3, 4 };
Array.Sort(oldNumbers);
```

Accéder aux données d'un Array

Les Éléments sont accessibles à partir de 0 jusqu'à N-1

Accéder à des éléments spécifiques

```
int[] oldNumbers = { 1, 2, 3, 4, 5 };
int number = oldNumbers[2];
// OR
object number = oldNumbers.GetValue(2);
```

Itérer sur tous les éléments

```
int[] oldNumbers = { 1, 2, 3, 4, 5 };
...
for (int i = 0; i < oldNumbers.Length; i++)
{
    int number= oldNumbers[i];
}
// OR
foreach (int number in oldNumbers) { ... }
```

Leçon 4: Utiliser des Instructions de Décision

- Utiliser des instructions If unidirectionnel
- Utiliser d’autres instructions If
- Utiliser des instructions If Multi-Résultats
- Utiliser l’instruction Switch
- Lignes directrices pour le choix d’une structure de décision

Utiliser des instructions If unidirectionnel

Syntaxe

```
if ([condition]) [code to execute]

// OR

if ([condition])
{
    [code to execute if condition is true]
}
```

Opérateurs Conditionnels:

OR represented by **||**

AND represented by **&&**

Exemple

```
if ((percent >= 0) && (percent <= 100))
{
    // Add code to execute if a is greater than 50 here.
}
```

Utiliser d’autres instructions avec If

Fournir un bloc de code supplémentaire à exécuter si la [condition] renvoie une valeur booléenne false

Exemple

```
if (a > 50)
{
    // Greater than 50 here
}
else
{
    // less than or equal to 50 here
}

// OR

string carColor = "green";

string response = (carColor == "red") ?
    "You have a red car" :
    "You do not have a red car";
```

Utiliser des instructions If Multi-Résultats

Vous pouvez combiner plusieurs instructions If pour créer une instruction multi-résultats

Exemple

```
if (a > 50)
{
    // Add code to execute if a is greater than 50 here.
}
else if (a > 10)
{
    // Add code to execute if a is greater than 10 and less than
    // or equal to 50 here.
}
else
{
    // Add code to execute if a is less than or equal to 50 here.
}
```

Utiliser l'instruction Switch

L'instruction switch vous permet d'exécuter un ou plusieurs blocs de code selon la valeur d'une variable ou d'une expression

Exemple

```
switch (a)
{
    case 0:
        // Executed if a is 0.
        break;

    case 1:
    case 2:
    case 3:
        // Executed if a is 1, 2, or 3.
        break;

    default:
        // Executed if a is any other value.
        break;
}
```

Lignes directrices pour le choix d'une structure de décision

Utiliser une structure **if** lorsque vous avez une condition unique qui contrôle l'exécution d'un seul bloc de code

Utiliser une structure **if/else** lorsque vous avez une condition unique qui contrôle l'exécution d'un oudeux blocs de code

Une structure **if/elseif/else** permet d'exécuter un ou plusieurs blocs de code en fonction des conditions qui impliquent plusieurs variables

Utiliser une structure **if** imbriqués pour effectuer une analyse plus complexe des conditions qui impliquent plusieurs variables

Utilisez une instruction **switch** pour effectuer une action fondée sur les valeurs possibles d'une seule variable

Leçon 5: Utiliser des Instructions d’Itération

- Types d’instructions d’iteration
- Utiliser l’instruction While
- Utiliser l’instruction Do
- Utiliser l’instruction For
- Instructions Break et Continue

Types d’instructions d’iteration

Les instructions d’Itération sont les suivantes:

while

Une boucle **while** vous permet d’exécuter un bloc de code zéro, une ou plusieurs fois

do

Une boucle **do** vous permet d’exécuter un bloc de code une ou plusieurs fois

for

Une boucle for vous permet d’exécuter du code un nombre défini de fois

Utiliser l'instruction While

La syntaxe d'une boucle while contient:

- Le mot clé **while** pour définir la boucle **while**
- Une condition qui est testée au démarrage de chaque itération
- Un bloc decode à éxécuter à chaque itération

Exemple

```
double balance = 100D;
double rate = 2.5D;
double targetBalance = 1000D;
int years = 0;
while (balance <= targetBalance)
{
    balance *= (rate / 100) + 1;
    years += 1;
}
```

Utiliser l’instruction Do

La syntaxe d’une boucle loop contient:

- Le mot clé **do** pour définir la boucle **do**
- Un bloc de codeà executer à chaque itération
- Une condition qui est testéeà la fin de chaque itération

Exemple

```
string userInput = "";
do
{
    userInput = GetUserInput();
    if (userInput.Length < 5)
    {
        // You must enter at least 5 characters.
    }
} while (userInput.Length < 5);
```

Utiliser l'instruction For

La syntaxe d'une boucle for contient:

- Le mot clé **for** pour définir la boucle **for**
- Les spécifications de la boucle (compteur, valeur de départ, limite, modificateur)
- Un bloc de code à executer à chaque instruction

Exemple

```
for (int i = 0; i < 10; i++)
{
    // Code to loop, which can use i.
}
```

Instructions Break et Continue

Instruction Break

```
while (oldNumbers.Length > count)
{
    if (oldNumbers[count] == 5)
    {
        break;
    }
    count++;
}
```

Permet de quitter la boucle et passer à la ligne suivante de code

Instruction Continue

```
while (oldNumbers.Length > count)
{
    if (oldNumbers[count] == 5)
    {
        continue;
    }
    // Code that won't be hit
    count++;
}
```

Permet d'ignorer le code restant dans l'itération actuelle, tester la condition puis démarrez la prochaine itération de la boucle

Lab: Using C# Programming Constructs

- Exercice 1: Bonjour c'est le week-end ...
- Exercice 2: Calculs en boucle
- Exercice 3: le jeu du plus ou du moins