C# and ASP.NET technoly - Ashraf KHABAR

Variables :

- We declare the variable like this :

```
int x ;
float a, b, c = 2.5 ;
char A ;
bool ok ;
```

- And constants like this :

```
const double PI = 3.14 ;
```

- Constatnt should always be initialized .
- We nee sometimes to convert variable into other (`casting`) :

```
int n = Int32.Parse(nbr.Text); //Converting into int
string m = n.ToString() //Converting into string
```

Arrays :

- We declare an `array` as :

```
type [] array = new Type[n] ;
```

With type = *int*, *double*, *string*, *Student* .... n = is an *int* indicate the size of the `array`.

- How to initialize the `array` :

- ```
  double[] marks = new double[] { 2, 2.6, 5.8 };
  double[] marks = { 2, 2.6, 5.8 };
  ```

- How to use it :

  ```
  for (int i = 0; i < marks.Lenght; i++) {

      // Your code
  }
  ```

- Bidimentianal `array` :

  ```
  double [, ] matrix = new double [, ] {
      {1, 4, 9},
      {6, 8, 2.5},
      {4, 5.9, 12.6}
  };
  ```

  And we use it like this :

  ```
  for (int i = 0; i < 3; i++)
  {
      for (int j = 0; j < 3; j++)
      {
          Console.WriteLine(matrix[i, j]);
      }
      Console.WriteLine();
  }
  ```

  `matrix.Lenght` = 9 because 3 rows x 3 columns .

OOP :

- We declare a `Class` in `C#` like this :

  ```
  public class Student {

      // Declaration of attributes
      private int Id ;
      private string name ;
      private int age ;

      // Declaration of methods
      public void display() {
  ```
  ```

```
        Console.WriteLine("Id : " + this.Id + " , Name : " + this.name + " ,
Age : " + this.age) ;
    }
}
```

Each `class` inside a `package` named `namespace` .

- `private` is the default parameter .
- `internal` : the acces is restricted to the assembly (projet or *dll* which have the class)
- We create the `object` like other lanuages :

```
Student student = new Student() ;
```

- *Properties* :

```
private int age ; // Private attribute

public int Age {
    get { return this.age} ;
    set { this.age = value}
}

// the Age plays the role of getters and setters .
```

- *Readonly* attributes : We want to deffine something `constant`, but we want to initilize it in the `Constructure` we use `readonly` :

```
public class Student {

    private readonly int age = 18;

    public Student (int value) {
        age = value ;
    }
}
```

- *Indexers* of class :

  The class *Student* :

```
class Student
{
    private int id;
```

```
    private string name;
    private int age;

    public Student(int id, string name, int age)
    {
        this.id = id;
        this.name = name;
        this.age = age;
    }

    public int Id { get => id; set => id = value; }
    public string Name { get => name; set => name = value; }
    public int Age { get => age; set => age = value; }
}
```

The class *School* :

```
class School
{
    private ArrayList list = new ArrayList();

    public Student this[int index]
    {
        get
        {
            return (Student)list[index];
        }

        set { list[index] = value; }
    }

    public void addToSchool(Student student)
    {
        list.Add(student);
    }
}
```

the class *Program* :

```
class Program
{
    static void Main(string[] args)
    {
        School ENSAT = new School();
        ENSAT.addToSchool(new Student(1, "Achraf KHABAR", 22));

        Console.WriteLine(ENSAT[0].Name);

        Console.ReadKey();
```

```
    }
}
```

Collections :

- `Collections` are specific `objects` .
- `Collections` are generic and non-generic .
- *ArrayList* :

```
ArrayList SomethingToStore = new ArrayList();
SomethingToStore.Add("Hello world");
SomethingToStore.Add(12);
SomethingToStore.Add(true);
SomethingToStore.Add(new Student(1, "Achraf KHABAR", 22));

foreach(Object obj in SomethingToStore)
{
    if (obj is Student)
        ((Student)obj).display();
    Console.WriteLine(obj);
}
```

We can use `Object` or `var` . To know if an `Object` is instance of a `class` we use `is` :

```
if (Obj is Student) {

    // In order to use the methods of Student we need to cast

    ((Student)obj).display();
}
```

- *Hashtable* :

```
Hashtable SomethingToStore = new Hashtable();
SomethingToStore.Add('1', "Ashraf");
SomethingToStore.Add(2, "KHABAR");

// We remoe based on the key :
SomethingToStore.Remove('1');

foreach( DictionaryEntry d in SomethingToStore)
{
    Console.WriteLine($"key : {d.Key} , Value : {d.Value}");
}
Console.ReadKey();
```

We can print on the console in tree ways in `C#` :

```
// way 1 :
Console.WriteLine("Key : " + d.Key + " , Value : " + d.Value) ;
// way 2 :
Console.WriteLine($"key : {d.Key} , Value : {d.Value}");
// way 3 :
Console.WriteLine("Key : {0}, Value : {1}", d.Key, d.Value);
```

- *SortedList* :

  This *Collection* is soted based on the `Key` :

```
SortedList colors = new SortedList();

colors.Add("r", "red");
    colors.Add("b", "blue");
colors.Add("g", "green");

foreach(DictionaryEntry color in colors)
{
    Console.WriteLine($"Key : {color.Key} , Value : {coloValue}");
}
```

  And we got in the `Output of console` :

```
Key : b , Value : blue
Key : g , Value : green
Key : r , Value : red
```

- *stack* and *queue* :

  - `Stack` : Last in first out
  - `Queue` : First in last out

```
Stack colors = new Stack();

colors.Push("red");
colors.Push("blue");
colors.Push("green");

colors.Pop();

foreach (object o in colors)
{
```

```
    Console.WriteLine(o);
}
```

We got on the `Console` :

```
blue
red
```

For the `Queue` :

```
Queue colors = new Queue();

colors.Enqueue("red");
colors.Enqueue("blue");
colors.Enqueue("green");

colors.Dequeue();

foreach (object o in colors)
{
    Console.WriteLine(o);
}
```

We got on the `Console` :

```
blue
green
```

- *List < T >* :

```
List<double> marks = new List<double>();
marks.Add(13.6);
marks.Add(11.6);
marks.Add(17.50);

foreach ( double mark in marks)
{
    Console.WriteLine(mark);
}
```

- *Dictionary < T >* :

```
Dictionary<int, string> SqlParams = new Dictionary<int, string>();

SqlParams.Add(1, "Id");
SqlParams.Add(2, "Name");
SqlParams.Add(3, "Age");

foreach( KeyValuePair<int, string> element in SqlParams)
{
    Console.WriteLine($"Key : {element.Key} , Value : {elemenValue}");
}
```

Overloading :

- Sometime we need to compare `Objects` not `Primitives` , so we need to `overload` some opperators in order to simplify the comparaison .
- Syntaxe :

```
public static return_type operator op (params) ;
```

- This example of class `Student` :

```
class Student
{
    private int id;
    private string name;
    private int age;
    private double[] marks;

    public Student(int id, string name, int age, int nbrModules)
    {
        this.id = id;
        this.name = name;
        this.age = age;
        this.marks = new double[nbrModules];
    }

    public int Id { get => id; set => id = value; }
    public string Name { get => name; set => name = value; }
    public int Age { get => age; set => age = value; }

    public double this[int index]
    {
        get
        {
            return this.marks[index];
        }
```

```
        set
        {
            this.marks[index] = value;
        }
    }

    public void display()
    {
        Console.WriteLine("Name : " + name);
    }

    public double meanOfStudent ()
    {
        double some = 0;
        if (this.marks == null)
        {
            Console.WriteLine("This student doesn't have any mark");
            return 0;
        } else
        {
            for (int i = 0; i < this.marks.Length; i++)
            {
                some += this.marks[i];
            }
            return some / this.marks.Length;
        }
    }
}
```

And we gonna add :

```
public static bool operator >(Student a, Student b)
{
    return a.meanOfStudent() > b.meanOfStudent() ;
}

public static bool operator <(Student a, Student b)
{
    return a.meanOfStudent() < b.meanOfStudent();
}
```

We cannot `overload` the operator "<" and not `overload` the ">" operator .

The `class` *Program* :

```
class Program
{
    static void Main(string[] args)
    {
```

```
        Student studentN1 = new Student(1, "ashraf KHABAR", 22, 6);
        Student studentN2 = new Student(2, "sami AOUAD", 22, 6);

        studentN1[0] = 12.6;
        studentN1[1] = 12.6;
        studentN1[2] = 12.6;

        studentN2[0] = 15.15;
        studentN2[1] = 15.15;
        studentN2[2] = 15.15;

        Console.WriteLine($"The mean of studen number 1 :
{studentN1.meanOfStudent()} , and the mean of the second student is
{studentN2.meanOfStudent()}");

        if (studentN1 > studentN2)
            Console.WriteLine("The student number one has better marks than
students number two .");
        else
            Console.WriteLine("The student number two has better marks than
students number one .");

        Console.ReadKey();
    }
}
```

We got in the `Output` on the `console` :

```
The mean of studen number 1 : 6.3 , and the mean of the second student is
7.575

The student number two has better marks than students number one .
```

Inheritance :

- The `sealed` class cannot be `Inherited` from other class .
- The `sealed` method cannot be `overrieded` by other subclasses .
- Syntaxe of `sealed` items :

```
public sealed class Name_of_class {
    ...
}
```

Or :

```
public sealed void Name_of_method(param1, param2, ...) {
    ...
}
```

- How to make `Inheritance` :

```
public class Name_of_mother_class {
    ...
    ...
    ...
}

public class Name_of_subclass : Name_of_mother_class {

    public Name_of_subclass() : base() {
        ...
        ...
    }

    ...
    ...
    ...
    ...
}
```

DAO.NET :

- DAO : (Data Access Object) is a technology which provides methods in order to access to the relational databases .
- Two mode of DAO :
    - Connected mode
    - Non connected mode
Data Provider	Description
System.Data.SqlClient	Specific to SQL Server
System.Data.OleDb	Handles data sources accessed via an OleDb driver
MySql.Data.MySqlClient	Specific to MySQL
Object	Description
Connection	Opens a connection to a specific data source
Command	Executes a command on a data source
DataReader	Reads a stream of data from a data source in connected mode. The access mode is read-only with a forward-only cursor.

Object	Description
DataSet	An object that resides in memory and corresponds to a local copy of data from a database. This data is written in XML and the schema is written in XSD.
DataAdapter	Acts as a liaison between a DataSet object and a data source. It fills a DataSet and reflects updates back to the data source.

-

Object	Code
Connection	*SqlConnection cnx = new SqlConnection(connection_string); cnx.Open();*
Command	*SqlCommand cmd = new SqlCommand(query, cnx); cmd.ExecuteNonQuery(); // Executes a command that doesn't return any rows cmd.ExecuteScalar(); // Retrieves a single value SqlDataReader dr = cmd.ExecuteReader(); // Used to retrieve a set of records and returns a DataReader while (dr.Read()) { // Display the value of the second column Console.WriteLine(dr.GetString(1)); } // Close the DataReader and Connection objects dr.Close(); cnx.Close();*

- How to execute a `sql` query using the `sql` :

  We have a table in database named *Etudiant* :

Id	Nom	Prenom	Vill	Age
1	Achraf	khabar	casablanca	22
2	Sami	aouad	casablanca	22

```
// Connection with the database using the connection string :
SqlConnection con = new SqlConnection("Data Source=
(localdb)\\MSSQLLocalDB;Initial Catalog=ecole;Integrated Security=True");

// oppening the connection with the database :
con.Open();

// Commande sql :
string sql = "select * from Etudiant";

// Link the sql query with database :
SqlCommand cmd = new SqlCommand(sql, con);

// Execute the comande :
SqlDataReader dr = cmd.ExecuteReader();

// Fetch the data :
while (dr.Read())
{
    Console.WriteLine($"Name : {dr.GetString(1)} , LastName
{dr.GetString(2)}");
}
```

We got in the `Output` :

```
Name : Achraf     , LastName : khabar
Name : Sami       , LastName : aouad
```

- *Stored Procedures* :

  The syntaxe :

```
CREATE PROCEDURE [dbo].Ajouter
    @param1 int ,
    @param2 char(20),
    @param3 char(20),
    @param4 char(20),
    @param5 int
AS
    insert into Etudiant values(@param1, @param2, @param3, @param4, @param5)
;
RETURN 0
```

  and the code of execution :

```
// Connection with the database using the connection string :
SqlConnection con = new SqlConnection("Data Source=
(locald\\MSSQLLocalDB;Initial Catalog=ecole;Integrated Security=True");

// oppening the connection with the database :
con.Open();

SqlCommand cmd = new SqlCommand("Ajouter", con);
cmd.CommandType = System.Data.CommandType.StoredProcedure;
SqlParameter p1 = new SqlParameter("param1", 3);
SqlParameter p2 = new SqlParameter("param2", "Oxford");
SqlParameter p3 = new SqlParameter("param3", "Maradona");
SqlParameter p4 = new SqlParameter("param4", "Tawajtat");
SqlParameter p5 = new SqlParameter("param5", 70);

cmd.Parameters.Add(p1);
cmd.Parameters.Add(p2);
cmd.Parameters.Add(p3);
cmd.Parameters.Add(p4);
cmd.Parameters.Add(p5);

cmd.ExecuteNonQuery();

Console.ReadKey();
```

- *Non connected mode* :

  How to load data in deconnected mode :

```
// Connection with the database using the connection string :
SqlConnection con = new SqlConnection("Data Source=
(locald\\MSSQLLocalDB;Initial Catalog=ecole;Integrated Security=True");

DataSet ds = new DataSet();
SqlDataAdapter da;

string sql = "select * from Etudiant;";
da = new SqlDataAdapter(sql, con);
da.FillSchema(ds, SchemaType.Source, "Etudiant");
da.Fill(ds, "Etudiant");
```

  How to Add :

```
DataRow row = result.Tables["[ecole].[dbo].[Etudiant]"].NewRow();
row["Id"] = 4;
row["Nom"] = "Arnold";
row["Prenom"] = "Xavier";
row["Ville"] = "Eldia";
row["Age"] = 35;

result.Tables["[ecole].[dbo].[Etudiant]"].Rows.Add(row);
```

LINQ :

- Language integrated Query are technologies that inetgrate using qury function withour `C#` .
- Syntaxe :

```
from [type] element in data_source
where [condition]
select element
```

- How to use it :

```
int[] data = new int[] { 12, 63, 41, 52, 10, 96, 22, 23 };

var query = from d in data
            select d;

foreach(int element in query)
{
```

```
    Console.WriteLine(element);
}
```

- How to filter the data :

```
int[] data = new int[] { 12, 63, 41, 52, 10, 96, 22, 23 };

var query = from d in data
            where d > 20
            select d;

foreach(int element in query)
{
    Console.WriteLine(element);
}
```

- Using the order by :

```
var query = from d in data
            where d > 20
            orderby a descending
            select d ;
```

- Agregation :

```
var query = from d in data
            where d > 20
            select d ;

// Count :
query.Count()
// Max :
query.Max()
// Min :
query.Min()
// Average :
query.Average()
// Frist element :
query.First()
```

- Avoiding repititions :

  We use `Distinct()` :

```
int[] data = new int[] { 12, 63, 41, 52, 10, 96, 22, 23 };
int[] data2 = new int[] { 16, 12, 10, 10, 63, 12, 15, 90 };

 var query = (from d in data
             from d2 in data2
             where d == d2
             select d).Distinct();

foreach(int element in query)
{
    Console.WriteLine(element);
}
```

- Joining :

```
int[] data = new int[] { 12, 63, 41, 52, 10, 96, 22, 23 };
int[] data2 = new int[] { 16, 12, 10, 10, 63, 12, 15, 90 };

 var query = (from d in data
             join d2 in data2
             on d equal d2
             select d).Distinct();

foreach(int element in query)
{
    Console.WriteLine(element);
}
```

- We can declare variable also :

  We use the `let` key word :

```
int[] data = new int[] { 12, 63, 41, 52, 10, 96, 22, 23 };
var query = from n int data
             let i = n%2
             where i == 1
             select n ;

// We will fetch only odd numbers .
```

- The key word `Into` :

```
int [] T = new int[]{1, 2, 3, 4, 5, 6, 7, 8, 9};
int [] A = new int[]{4, 5, 3, 3, 7, 8, 9};

var x =
```

```
(
    from int M in
    (
        from res in T
        where res < 5
        select res
    )
    where M > 2
    select M into prod
    from c in A
    where prod == c
    select c
).Distinct();
```

- Using *Collections* :

  We can store data in collection, store also `objects` in the collections, we have a `class` *Student*, with *Id*, *Name*, *Age*, and *Mean_of_marks* .

```
List<Student> school = new List<Student>();
school.Add(new student(1, "ashraf khabar", 22, 15));
school.Add(new student(2, "sami aouad", 22, 18));
school.Add(new student(3, "karim idrissi", 22, 19));

// extracting data :

var query = from S in school
            where S.Mean_of_marks >= 16
            orderby S.Name ascending
            select S;

foreach(Student St in query) {
    Console.WriteLine($"Name : {St.Name} , Mean of marks :
{St.Mean_of_marks}");
}
```

- Using *generic objects* like `Java script` :

  We can generate generic object like we do in `Java script` :

```
var Object = new[] {

    new {id=1, Name="Ashraf khabar", Age=22, Mane_of_marks=15},
    new {id=2, Name="Sami aouad", Age=22, Mane_of_marks=18},
    new {id=3, Name="Karim idrissi", Age=22, Mane_of_marks=19},
};

var query = (
```

```
    from obj in Object
    select obj.Name
).Distinct().Count()
```

- How to *Link* with data source from *Database* :
    - We gonna use a new *Item* named `Link to SQL Classes` .
    - We gonna create a new file with `dbml`, for example *Link.dbml* .
- How we use this file :

  We already created a file named *Link.dbml*, so we gonna create an instance from `LinkDataContext` .

```
{name_of_file}DataContext name_of_instance = new
{name_of_file}DataContext();
```

  For example :

```
LinkDataContext data = new LinkDataContext();
```

- How we can add a new row in the table :

```
LinkDataContext data = new LinkDataContext();

Etudiant student = new Etudiant { Id = 7, Nom = "OS", Prenom = "SYSTEM", Age
= 70, Ville = "Manhaten" };

data.Etudiants.InsertOnSubmit(student);
data.SubmitChanges();
```

  Etudiant is not a name of a `class` in my project, but it's the name of the table where i want to store the data, and it's the same name of table I added in the *Link.dnml* file .

  other example :

```
LinkDataContext data = new LinkDataContext();

user user = new user { Id = 12, login = "Ashraf khabar", pswd = "12345678",
connected = "True" };

data.users.InsertOnSubmit(user);
data.SubmitChanges();
```

- How to *Update* a row :

```
LinkDataContext data = new LinkDataContext();

var E = (from etudiant in data.Etudiants
         where etudiant.Id == 2
         select etudiant).FirstOrDefault();

E.Nom = "Alfredo";
E.Prenom = "Gastro";
E.Ville = "NewYork";

data.SubmitChanges();
```

- How to *Delete* a row :

```
LinkDataContext data = new LinkDataContext();

var E = (from etudiant in data.Etudiants
         where etudiant.Id == 2
         select etudiant).FirstOrDefault();

data.Etudiants.DeleteOnSubmit(E);

data.SubmitChanges();
```

- *Selection* :

```
LinkDataContext data = new LinkDataContext();

var E = (from etudiant in data.Etudiants
         where etudiant.Id == 2
         select etudiant).FirstOrDefault();
```

- *Fetch more than one element* :

```
LinkDataContext data = new LinkDataContext();

var E = (from etudiant in data.Etudiants
         select etudiant);

foreach (Etudiant e in E)
{
    Console.WriteLine($"{e.Id}, {e.Nom}");
}
```

- How to make *Joining* :

```
LinkDataContext data = new LinkDataContext();

var query = (from student in data.students
                join mean in data.means on student.student_iequals
mean.student_id
                where mean.mean_score > 16
                select student.student_name).Distinct();

foreach(string s in query)
{
    Console.WriteLine($"{s}");
}
```

- Multiple joining statements :

  If you want to use the same `Schema` you can exexute the following *Script* :

```
CREATE TABLE student (
student_id INT PRIMARY KEY,
student_name VARCHAR(50) NOT NULL,
student_age INT NOT NULL,
student_gender VARCHAR(10) NOT NULL,
student_major VARCHAR(50) NOT NULL
);

CREATE TABLE mean (
mean_id INT PRIMARY KEY,
student_id INT NOT NULL,
subject VARCHAR(50) NOT NULL,
mean_score FLOAT NOT NULL,
FOREIGN KEY (student_id) REFERENCES student(student_id)
);

CREATE TABLE branches (
branch_id INT PRIMARY KEY,
branch_name VARCHAR(50) NOT NULL
);

ALTER TABLE student
ADD branch_id INT NOT NULL;

ALTER TABLE student
ADD FOREIGN KEY (branch_id) REFERENCES branches(branch_id);

-- Insert data into the "branches" table
INSERT INTO branches (branch_id, branch_name) VALUES (1, 'GINF');
INSERT INTO branches (branch_id, branch_name) VALUES (2, 'UNINET');
INSERT INTO branches (branch_id, branch_name) VALUES (3, 'Central');

-- Insert data into the "student" table
```

```
INSERT INTO student (student_id, student_name, student_age, student_gender,
student_major, branch_id)
VALUES (1, 'Achraf khabar', 22, 'M', 'GINF2', 1);

INSERT INTO student (student_id, student_name, student_age, student_gender,
student_major, branch_id)
VALUES (2, 'Sami aouad', 22, 'M', 'UNINET', 2);

INSERT INTO student (student_id, student_name, student_age, student_gender,
student_major, branch_id)
VALUES (3, 'Karim indrissi', 22, 'M', 'Central', 3);

-- Insert data into the "mean" table
INSERT INTO mean (mean_id, student_id, subject, mean_score)
VALUES (1, 1, 'Java', 15);

INSERT INTO mean (mean_id, student_id, subject, mean_score)
VALUES (2, 1, 'XML', 14);

INSERT INTO mean (mean_id, student_id, subject, mean_score)
VALUES (3, 2, 'DS with C', 20);

INSERT INTO mean (mean_id, student_id, subject, mean_score)
VALUES (4, 2, 'java', 19);

INSERT INTO mean (mean_id, student_id, subject, mean_score)
VALUES (5, 3, 'Cpp', 20);

INSERT INTO mean (mean_id, student_id, subject, mean_score)
VALUES (6, 3, 'java', 20);
```

The C# code :

```
LinkDataContext data = new LinkDataContext();

Table<student> students = data.GetTable<student>();
Table<mean> means = data.GetTable<mean>();
Table<branch> branches = data.GetTable<branch>();

var query = (from student in students
             join mean in means
             on student.student_id equals mean.student_id
             join branch in branches
             on student.branch_id equals branch.branch_id
             where branch.branch_name == "GINF"
             select student).Distinct();

foreach(student student in query)
{
    Console.WriteLine($"{student.student_name}");
}
```

Management of XML files using C# :

- We create an instance of *XDocument* .
- We need to specify the location of the `XML` file .
- First we neet : `using System.Xml.Linq;` on the header :

```
XDocument doc = new XDocument();
XElement racine;

try {
    doc = XDocument.Load(@"c:\documents\doc.XML");
} catch (Exception ex) {
    doc = new XDocument(new XDeclaration("1.0", "uft-8", "no"),
                        new XElement("ConnectionWithDb"));
}

root = doc.Root;
root.Add(
    new XElement(
        "ConnectionString",
        new XElement("Host", "localhost"),
        new XElement("user", "root"),
        new XElement("password", ""),
        new XElement("dbName", "myDatabase")
    )
);

doc.Save(@"c:\documents\doc.XML");
```

We got an `XML` like this :

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<ConnectionWithDb>
    <ConnectionString>
        <Host>localhost</Host>
        <user>root</user>
        <password></password>
        <dbName>myDatabase</dbName>
    </ConnectionString>
</ConnectionWithDb>
```

- How add an element in the XML file :

```
XDocument doc = new XDocument();

doc =
XDocument.Load(@"C:\Users\ashraf\Desktop\CSHARP\testing\testing\DbsettinXML"
);
XElement root = doc.Root;

root.Add(
    new XElement(
          "DataBaseType",
        new XElement("type", "MySQL")
    )
);
doc.Save(@"c:\documents\doc.XML");
```

We got a file like this :

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<ConnectionWithDb>
    <ConnectionString>
        <Host>localhost</Host>
        <user>root</user>
        <password></password>
        <dbName>myDatabase</dbName>
    </ConnectionString>
    <DataBaseType>
        <type>MySQL</type>
    </DataBaseType>
</ConnectionWithDb>
```

- How to fetch data from the `XML` file :

```
XDocument doc = XDocument.Load("@c:\documents\doc.XML");

XElement root = doc.Root;

string host = "";
string user = "";
string password = "";
string dbName = "";
string dataBaseType = "";

var elements = root.Elements("ConnectionString");
Console.WriteLine(elements.Count());
foreach (XElement element in elements)
{
    host = element.Element("Host").Value;
    user = element.Element("user").Value;
    password = element.Element("password").Value;
```

```
    dbName = element.Element("dbName").Value;
    break;
}

var dbType = root.Elements("DataBaseType");
foreach(XElement element in dbType)
{
    dataBaseType = element.Element("type").Value;
}

Console.WriteLine(host);
Console.WriteLine(dataBaseType);
```

quick reminder :

How to use the DAO inside a *C#* code :

- Execution of normal selection :

```
SqlConnection con = new SqlConnection("Data Source=
(localdb)\\MSSQLLocalDB;Initial Catalog=ecole;Integrated Security=True");

con.Open();


string query = "select * from Student";

SqlCommand cmd = new SqlCommand(query, con);

SqlDataReader dr = cmd.ExecuteReader();

while(dr.Read())
{
    Console.WriteLine($"Nmae : {dr.GetString(1)}");
}
```

- Executing of inserting :

```
SqlConnection con = new SqlConnection("Data Source=
(localdb)\\MSSQLLocalDB;Initial Catalog=ecole;Integrated Security=True");

con.Open();

string query = "Insert into student values (4, 'Yassir Amami', 22, 'M',
'GINF2', 1)";

SqlCommand cmd = new SqlCommand(query, con);
```

```
cmd.ExecuteNonQuery();
```

- Executing queries that return a single value :

```
SqlConnection con = new SqlConnection("Data Source=
(localdb)\\MSSQLLocalDB;Initial Catalog=ecole;Integrated Security=True");

con.Open();

string query = "select count(*) from student";

SqlCommand cmd = new SqlCommand(query, con);

string result = cmd.ExecuteScalar().ToString();

int nbrOfSyudent = Int16.Parse(result);

Console.WriteLine($"The number of student is : {nbrOfSyudent}");
```

- Execute a procedure of selection :

  The Procedure :

```
CREATE PROCEDURE [dbo].GINF1Stuent
AS
    SELECT student_name, student_age, student_gender from student where
student_major = 'GINF2'
RETURN 0
```

  Execution :

```
SqlConnection con = new SqlConnection("Data Source=
(localdb)\\MSSQLLocalDB;Initial Catalog=ecole;Integrated Security=True");

con.Open();

SqlCommand cmd = new SqlCommand("GINF1Stuent", con);

cmd.CommandType = System.Data.CommandType.StoredProcedure;

SqlDataReader dr = cmd.ExecuteReader();

while (dr.Read())
{
    Console.WriteLine($"Name : {dr.GetString(0)} , Age : {dr.GetInt32(1)} ,
Gender : {dr.GetString(2)}");
```

```
}
```

- Execute a procedure of Inserting :

  The procedure :

```
CREATE PROCEDURE [dbo].AddStudent
@param1 int ,
@param2 varchar(20),
@param3 int,
@param4 varchar(20),
@param5 varchar(20),
@param6 int
AS
    insert into student values(@param1, @param2, @param3, @param4, @param5,
@param6) ;
RETURN 0
```

  The execution :

```
SqlConnection con = new SqlConnection("Data Source=
(localdb)\\MSSQLLocalDB;Initial Catalog=ecole;Integrated Security=True");

con.Open();
SqlCommand cmd = new SqlCommand("AddStudent", con);
cmd.CommandType = System.Data.CommandType.StoredProcedure

SqlParameter p1 = new SqlParameter("param1", 5);
SqlParameter p2 = new SqlParameter("param2", "OussamGuemmar");
SqlParameter p3 = new SqlParameter("param3", 22);
SqlParameter p4 = new SqlParameter("param4", "M");
SqlParameter p5 = new SqlParameter("param5", "GINF2");
SqlParameter p6 = new SqlParameter("param6", 1)

cmd.Parameters.Add(p1);
cmd.Parameters.Add(p2);
cmd.Parameters.Add(p3);
cmd.Parameters.Add(p4);
cmd.Parameters.Add(p5);
cmd.Parameters.Add(p6)
cmd.ExecuteNonQuery();
```