



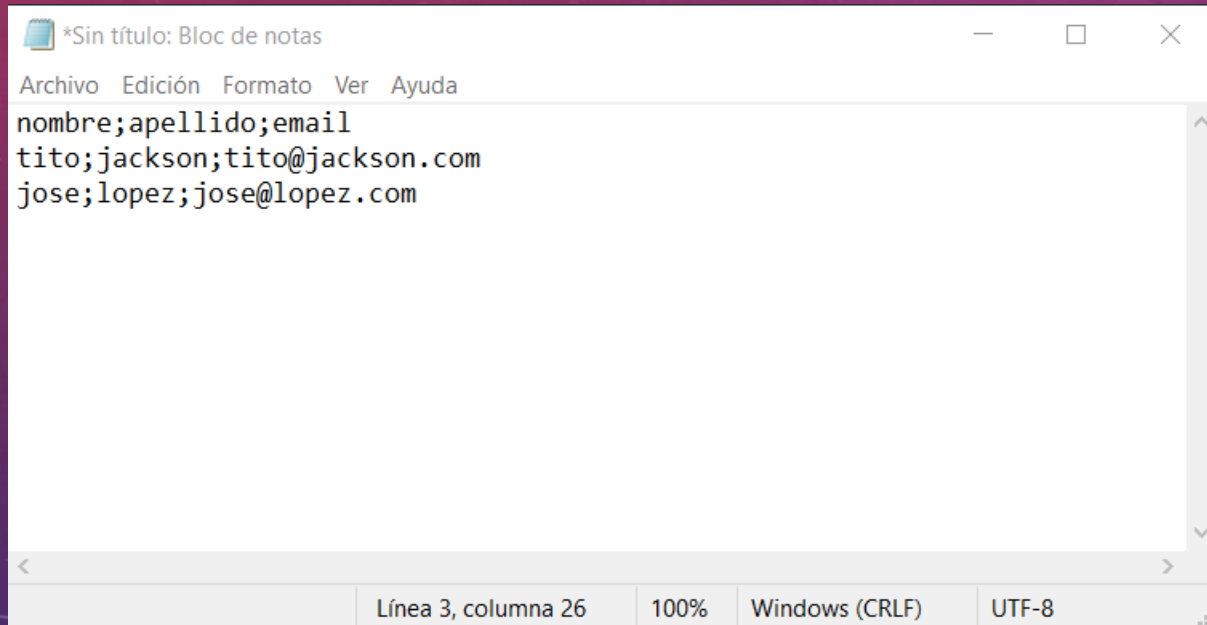
PERSISTENCIA

ARCHIVOS CSV

- Son archivos de texto cuyos valores están separados por comas o punto y comas.
- Los valores en el archivo van armando una separación en filas y columnas.
- Pueden armarse con gran facilidad en varios lenguajes y son más livianos que un archivo xls por ejemplo.
- Sirven para manejar gran cantidad de datos con un relativo bajo costo.
- Se utilizan mucho para exportar datos o manejar bases de datos.

INTEROPERABILIDAD DE UN CSV

- Podemos crear un archivo csv muy fácilmente con el bloc de notas o cualquier editor de texto.



A screenshot of a Windows Notepad window titled "*Sin título: Bloc de notas". The window contains three lines of text separated by semicolons, representing a CSV file. The status bar at the bottom indicates "Línea 3, columna 26", "100%", "Windows (CRLF)", and "UTF-8".

```
*Sin título: Bloc de notas
Archivo Edición Formato Ver Ayuda
nombre;apellido;email
tito;jackson;tito@jackson.com
jose;lopez;jose@lopez.com
Línea 3, columna 26 100% Windows (CRLF) UTF-8
```

	A	B	C	D
1	nombre	apellido	email	
2	tito	jackson	tito@jackson.com	
3	jose	lopez	jose@lopez.com	
4				
5				
6				
7				

XML

- Es un lenguaje de etiquetado que nos permite almacenar y estructurar datos de forma legible.
- Se utiliza mucho como forma standarizada de comunicar varias aplicaciones entre sí.
- Se utiliza para bases de datos, APIs, etcétera.
- Permite la conservación e integridad de los datos ya que permite enviar la información junto con su descripción en un mismo archivo.
- Usa un formato de etiquetado similar a HTML

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Personas>
  <Persona>
    <nombre>tito</nombre>
    <apellido>jackson</apellido>
    <email>tito@jackson.com</email>
  </Persona>
  <Persona>
    <nombre>jose</nombre>
    <apellido>lopez</apellido>
    <email>jose@lopez.com</email>
  </Persona>
</Personas>
```

JSON

- JSON es un formato de intercambio de información basado en la notación de JavaScript.
- Se utiliza más comúnmente en desarrollo web y, obviamente, en JavaScript.
- Es un formato fácilmente legible para computadoras al igual que XML
- Suele utilizarse para hacer solicitudes y respuestas a APIs
- Pueden ver más en <https://www.json.org/json-en.html>

JSON

```
{  
  "Personas": [  
    {  
      "nombre": "tito",  
      "apellido": "jackson",  
      "email": "tito@jackson.com"  
    },  
    {  
      "nombre": "jose",  
      "apellido": "lopez",  
      "email": "jose@lopez.com"  
    }  
  ]  
}
```


CREANDO ARCHIVOS DESDE C#

- A diferencia de lo que sucede en C, el garbage collector se encarga también de cerrar archivos que puedan haber llegado a quedar abiertos.
- Hay distintas clases de .NET que nos van a ayudar a leer y escribir archivos.
- Los modos de lectura y escritura serán los mismos que en C (read, write, append).

ESCRIBIENDO EN ARCHIVOS CSV

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Threading.Tasks;  
using System.Text;  
using System.Xml;  
using System.IO;
```

```
public string guardarEnArchivo(string archivo)  
{  
    string separador = ";";  
    StringBuilder salida = new StringBuilder();  
    String[] encabezados = { "nombre", "apellido", "email" };  
    salida.AppendLine(string.Join(separador, encabezados));  
    foreach (Persona persona in listadoPersonas)  
    {  
        string[] linea = { persona.nombre, persona.apellido, persona.email };  
        salida.AppendLine(string.Join(separador, linea));  
    }  
    try  
    {  
        File.WriteAllText(archivo, salida.ToString());  
        return "ok";  
    }  
    catch (Exception ex)  
    {  
        return ex.Message;  
    }  
}
```

ALGUNOS CONCEPTOS

- **StringBuilder:** Nos permite construir cadenas de caracteres dinámicamente y de forma eficiente. A diferencia de un String, no es una cadena como un string si no que su manejo es más eficiente en memoria y permite el agregado dinámico de líneas o generación de resultados.
 - Con el método Append podemos insertar data al final
- **File:** Proporciona algunos métodos estáticos para leer y escribir información en archivos.

BLOQUES TRY CATCH

- Cuando manejamos archivos o tratamos de acceder a distintos elementos ajenos a nuestro programa pueden aparecer varias excepciones. Por ejemplo, podemos no tener acceso al archivo que queremos leer o escribir o puede no existir.
- Para que el programa no explote en mil pedazos podemos “atrapar” esas excepciones y manejarlas.

BLOQUES TRY Y CATCH

```
try
{
    //acá van todas las instrucciones que pueden hacer que nuestro programa explote
}
catch(Exception ex)
{
    //acá va lo que queremos que nuestro programa haga en caso de que explote
    //Por lo general podemos revertir algún cambio hecho a medias o informar el error
}
finally
{
    //codigo que se va a ejecutar siempre luego de una secuencia try y, si sucede, una secuencia catch
}
```


LEYENDO UN CSV

```
public string leerCSV(string archivo)
{
    try
    {
        using (StreamReader reader = new StreamReader(archivo))
        {
            string encabezado = reader.ReadLine(); // Leer la línea de encabezado si existe

            while (!reader.EndOfStream)
            {
                string linea = reader.ReadLine();
                string[] valores = linea.Split(';'); // Separar los valores por el separador adecuado
                if (valores.Length == 3)
                {
                    string nombre = valores[0];
                    string apellido = valores[1];
                    string email = valores[2];
                    this.crearPersona(nombre, apellido, email)
                }
            }

            return "ok";
        }
    }
    catch (Exception ex)
    {
        return ex.Message;
    }
}
```

ALGUNOS CONCEPTOS

- **StreamReader:** Es una clase que se utiliza para leer caracteres de una secuencia de entrada, como un archivo o un flujo de datos. Proporciona métodos convenientes para leer texto de manera eficiente.
- **Using:** Es un bloque en C# que permite garantizar que cualquier recurso se libere después de su uso. Por ejemplo, StreamReader implementa la interfaz IDisposable que implica que un elemento es “Descartable”. Puede usarse para leer archivos, conectarse a base de datos o flujos de red.
- **Split:** Es un método que permite separar un string en un vector de strings teniendo un determinado caracter de separación.

ESCRIBIENDO EN XML

```
public string guardarEnXML(string archivo)
{
    try
    {
        XmlDocument xmlDoc = new XmlDocument();
        XmlDeclaration xmlDeclaration = xmlDoc.CreateXmlDeclaration("1.0", "UTF-8", null);
        xmlDoc.AppendChild(xmlDeclaration);

        XmlElement raiz = xmlDoc.CreateElement("Personas");
        xmlDoc.AppendChild(raiz);

        foreach (Cuenta cuenta in listadoCuentas)
        {
            XmlElement elemento = xmlDoc.CreateElement("Persona");

            XmlElement nombreElement = xmlDoc.CreateElement("Nombre");
            nombreElement.InnerText = persona.getNombre();
            elemento.AppendChild(nombreElement);

            XmlElement apellidoElement = xmlDoc.CreateElement("Apellido");
            apellidoElement.InnerText = persona.getApellido();
            elemento.AppendChild(apellidoElement);

            XmlElement emailElement = xmlDoc.CreateElement("Email");
            nombreElement.InnerText = persona.getEmail();
            elemento.AppendChild(emailElement);

            raiz.AppendChild(elemento);
        }

        xmlDoc.Save(archivo);
        return "ok";
    }
    catch (Exception ex)
    {
        return ex.Message;
    }
}
```

LEYENDO UN XML

```
public string leerXML(string archivo)
{
    try
    {
        XmlDocument xmlDoc = new XmlDocument();
        xmlDoc.Load(archivo);

        XmlNodeList nodos = xmlDoc.SelectNodes("//Persona");

        foreach (XmlNode nodo in nodos)
        {
            ulong CBU = System.Convert.ToUInt64(nodo.SelectSingleNode("Nombre").InnerText);
            string cliente = nodo.SelectSingleNode("Apellido").InnerText;
            float saldo = nodo.SelectSingleNode("Email").InnerText;

            this.crearCuenta(CBU, cliente, saldo);
        }

        return "ok";
    }
    catch (Exception ex)
    {
        return ex.Message;
    }
}
```