ARQUITECTURA SOA.

Service Oriented Architecture – Arquitectura Orientada a Servicios.

- Principios o tipo de diseño de software pensado para desarrollar sistemas distribuidos orientados a servicios cuya implementación se realiza básicamente a través de servicios web.

Servicio.

- Es una unidad autónoma de software con una o varias funciones pensadas para realizar una unidad de trabajo o tarea determinada (comprobar un saldo, hacer un cálculo, identificar a un usuario, consultar el tiempo que va a hacer) para el consumidor y que esperan la siguiente solicitud.
- Estas unidades conforman componentes distribuidos y reutilizables a través de Internet.
- En realidad, un Servicio Web (o componente), es una colección de procedimientos (métodos) a los que llamar desde cualquier lugar, por ejemplo, desde Internet. Se puede utilizar para autenticar usuarios, validar tarjetas de crédito, calcular precios, etc.

SERVICIOS WEB – WEB SERVICES.

- **Concepto.**
  - Sistema de comunicación, protocolos y estándares que usan dos dispositivos electrónicos conectados a la misma red, para intercambiar datos entre sistemas o aplicaciones.
  - Software para la comunicación e interoperabilidad entre dispositivos.
  - Interfaces a través de las cuales un dispositivo puede hacer uso del servicio de un servidor.
  - Se encarga de permitir la transmisión de solicitudes y respuestas entre diferentes servidores o aplicaciones, sin importar las diferencias existentes entre los lenguajes de programación en el que fueron desarrolladas o la plataforma en la que se ejecutan.
  -
- **Tecnología.**
  - La tecnología de los servicios web se caracteriza por ser:
    - **Multiplataforma:**
      - Cliente y servidor no tienen por qué contar con la misma configuración para comunicarse. El servicio web se encarga de hacerlo posible.
    - **Distribuida:**
      - Los recursos del servicio web pueden estar repartidos en equipos interconectados, sin embargo, el usuario los percibe como un todo homogéneo.
      - Siguen la tendencia de modularizar las aplicaciones, así se pueden crear componentes distribuidos y reutilizables a través de Internet.

- Esto permitirá utilizar estos servicios desde distintos clientes, como ordenadores, teléfonos móviles, PDA's, etc., independientemente de la interfaz de estos clientes, además de poder reutilizar dichos servicios desde otros servidores, lo cual nos permite desarrollar aplicaciones distribuidas en Internet.

- **Características:**
  - Son accesibles vía Web.
    - Para ello debe utilizan protocolos de transporte estándares como HTTP, y codifican sus mensajes en un lenguaje estándar conocido por cualquier cliente que quiera utilizar el servicio.
    - Un servicio web queda referenciado y localizado mediante una URL.
  - Contienen una descripción de sí mismos.
    - Así una aplicación puede saber cuál es su función y utilidad.
    - La aplicación debe saber también cuál es su interfaz para utilizarlo de forma automática, sin la intervención del usuario.
  - Están localizados.
    - Hay mecanismo que permiten a una aplicación encontrar automáticamente el Servicio Web que se necesita o que realiza una determinada función.
- **XML.**
  - Elemento clave en el sistema de servicios web.
  - Es la manera en que están codificados los datos para que se pueden intercambiar y procesar por diferentes sistemas.
  - XML permite que se comuniquen diferentes dispositivos, aunque las aplicaciones o sistemas usen lenguajes diferentes de programación.
- **Ejemplos.**
  - **Muchas interacciones cotidianas entre aplicaciones usan servicios web:**
    - Conectar la información de una cuenta de Facebook con un juego descargado.
    - Utilizar la información de inicio de sesión de Google.
    - Abrir una nueva cuenta en otra aplicación sin rellenar el formulario.
    - Comprobar el saldo de un cliente.
    - Procesamiento de pagos de terceros.
    - Un vendedor de lo que sea, por ejemplo, de material de una ferretería, realiza búsqueda de herramientas, componentes o piezas en los inventarios de diversos proveedores.
    - Compras de billetes de avión, tren, etc. y reservas de viajes.
    - Identificación de un cliente por una empresa independientemente de que contacte con ella mediante móvil, Tablet u ordenador.
    - Utilización de aplicaciones de terceros dentro de una página web como aplicaciones meteorológicas, de tráfico, índices bursátiles, mapas, contenidos para compras de tiendas virtuales, etc.
    - Amazon Web Services.
    - API web de Google.
- **Roles y funciones de los servicios web.**

- El esquema de funcionamiento de los servicios web, requiere de tres elementos fundamentales:
    - **Proveedor del servicio web.**
        - Es quien lo diseña, desarrolla e implementa.
        - También es quien lo hace disponible para su uso, dentro de una organización o de forma pública.
    - **Consumidor del servicio.**
        - Es quien accede al componente para utilizar los servicios que presta.
    - **Agente de servicio.**
        - Es el enlace entre proveedor y consumidor para efectos de publicación, búsqueda y localización del servicio.

API

- Interfaz de programación de aplicaciones (API).
- Conjunto de estándares, requisitos o instrucciones que permite que una aplicación preste servicio a otra aplicación, dispositivo o plataforma para un mejor uso del software o la aplicación.
- Usando una API, varios programas se comunican entre sí.

- **Tipos de servicios Web:**
    - SOAP.
    - RESTful.
    - GraphQL.

SOAP. (SIMPLE OBJECT ACCESS PROTOCOL - PROTOCOLO SIMPLE DE ACCESO A OBJETOS).

- **Concepto y características**.
    - Es un protocolo de comunicación entre aplicaciones.
    - Usa el protocolo HTTP para realizar peticiones de datos y recursos. Los mensajes SOAP se transportan usando este protocolo.
    - Es un formato para enviar y recibir mensajes, que permite implementar servicios web dentro de una arquitectura orientada a servicios (SOA).
    - Proporciona una manera de comunicarse entre aplicaciones que se ejecutan en sistemas operativos diferentes, con diferentes tecnologías y lenguajes de programación.
    - Es independiente de la plataforma.
    - Se basa en XML estándar.
    - Es una recomendación W3C.
- **Estructura de un mensaje SOAP.**
    - **Sobre (Envelope).**
        - Es el contendedor o elemento raíz de un mensaje SOAP.
        - Contiene el mensaje SOAP y es similar a un sobre ordinario.
        - Es obligatorio.
    - **Encabezado (Header).**

        - Contiene información de la aplicación como las características de autenticación, direccionamiento y enrutamiento.
        - Es opcional.
    - **Cuerpo (Body).**
        - Contiene el mensaje de la aplicación que se está transportando e incluye:
            - Operación remota que se invoca.
            - Datos (parámetros), que se intercambian.
        - Es obligatorio.
- **Ejemplo de mensaje SOAP.**

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <SOAP-ENV:Body>
                <ns:getTemperatura xmlns:ns="http://meteo.es/ns">
                        <area>Sevilla</area>
                </ns:getTemperatura>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

WSDL

- Web Services Description Language – Lenguaje de Descripción de un Servicio web.
- Está basado en XML.
- Describe un servicio Web, de manera que a través de un documento WSDL, se suministra la información necesaria para que el cliente interactúe con el servicio web.
- WSDL es extensible y se puede utilizar para describir, prácticamente, cualquier servicio de red.
- WSDL proporciona una descripción entendible que indica cómo se debe llamar al servicio, qué parámetros espera, y qué estructuras de datos devuelve.
- Incluye tipos y estructuras de datos que se van a usar, formatos, mensajes URL, etc.

UDDI

- Permite localizar Servicios Web.
- Define la especificación para construir un directorio distribuido de Servicios Web, donde los datos se almacenan en XML.
- En este registro se almacena información sobre servicios y sobre las organizaciones que los proporcionan, la categoría en la que se encuentran, y sus instrucciones de uso normalmente en WSDL.
- Define una API para trabajar con dicho registro, que nos permitirá buscar datos almacenados en él, y publicar datos nuevos.
- Tipos de información guardada en un directorio UDDI.
    - **Páginas blancas.**
        - Contine datos de las organizaciones (dirección, información de contacto, etc.).
    - **Páginas amarillas**.

    - Contine una clasificación de las organizaciones (según tipo de industria, zona geográfica, etc.).
- **Páginas verdes**.
    - Contine información técnica sobre los servicios que se ofrecen.
    - Incluye las instrucciones para utilizar los servicios.
    - Estas instrucciones, interesa que se especifiquen de forma estándar mediante un documento WSDL.

REST.

- **RE**presentational **S**tate **T**ransfer
- Otro tipo de tecnología que nos permita realizar una API con Web Services.
- Se diferencia de SOAP, en que más un protocolo, es una definición de arquitectura que indica cómo realizar la transferencia y manejo de datos a través de servicios web, por lo que no está estructurado bajo estándares definidos.
- Es más ligero.
- Funciona con XML, JSON y otros.
- Los servicios web basados en REST se denominan RESTful Web Services.

GraphQL.

- Lenguaje de consulta y un entorno de ejecución del lado del servidor para API's.
- Al igual que SOAP y REST, usa HTTP.
- Permite realizar solicitudes para obtener datos de múltiples fuentes con una única llamada a la API.
- Desarrollado por Facebook, paso a ser Open Source a partir del 2015.

Herramientas de desarrollo.

- C# o ASP.NET incluida en Visual Studio para la plataforma .NET.
- Java y sus extensiones:
    - JAW-WS para servicios web SOAP.
    - JAW-RS para servicios web REST.
    - Con Java se pueden usar IDE's como NetBeans o Eclipse.

Herramientas de prueba de servicios web.

- SoapUI.
- SoapSonar.
- Wizdler.
- WebInject
- Stylus Studio.

PROGRAMACIÓN EN C#

Características del lenguaje.

- Es un lenguaje orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET.
- **Versiones.**

- C#8 (septiembre 2019).
- C#9 (finales 2020).
- C#10 (marzo 2022).
- C#12 (noviembre 2023).
- C#13 (mayo 2024).

- **Código fuente**.
  - Los archivos con código fuente se guardan con la extensión .cs.
  - Estos archivos hay que compilarlos para poder crear un archivo ejecutable.
  - Se usa el compilador csc.exe, que está ubicado en la siguiente ruta de acceso: C:\Windows\Microsoft.NET\Framework64\v4.0.30319\
  - Para que el compilador funcione en cualquier punto la ruta anterior hay que incluirla en la variable de sistema *path*.
- **Compilación en consola**:
  - Acceder a la carpeta que contiene el archivo fuente, es decir, el archivo con la extensión *.cs,* y usar la instrucción csc.exe.
  - Sintaxis:
    - csc.exe nombre del archivo a compilar.cs
    - Aparecerá un archivo con el mismo nombre que el archivo fuente, pero con la extensión *.exe*.
  - Ejemplo:
    - C:/>csc.exe bucles.cs
    - Esto creará el archivo bucles.exe
- **Ejecutar en consola un archivo compilado**.
  - Acceder a la carpeta que contiene el archivo ejecutable, es decir, el archivo con la extensión *.exe.*
  - *Ejecutar un archivo compilado:*
    - En CMD.
      - Escribir el nombre del archivo sin la extensión para ejecutarlo.
      - Sintaxis:
        - C:/ruta de acceso/nombre del archivo sin la extensión.
    - En la terminal de Visual Studio Code.
      - Sintaxis:
        - ./nombre del archivo sin la extensión.
        - ./ruta de acceso/nombre del archivo sin la extensión.
    - Ejemplos:
      - C:/>bucles
      - ./bucles
      - ./ejercicios/bucles
- **Editar Código en C#.**
  - Editores de texto.
    - Bloc de notas de Windows, por ejemplo.
    - El código fuente luego hay que compilarlo con el compilador de línea de comando C#, csc.exe, que viene con .NET.

- Editores de código.
  - Notepad++, Visual Studio Code y otros.
  - Al igual que con la anterior opción, pueden crearse los archivos ejecutables usando el compilador csc.exe.
- IDE (Entorno de Desarrollo Integrado).
  - Al ser C# un producto de Microsoft, éste lo incluye en Visual Studio junto con otros lenguajes y herramientas de desarrollo web como ASP.NET, Visual Basic .NET, C++, PHP, JavaScript y otros.
  - Visual Studio (última versión 2022 v.17.10.3), permite crear y ejecutar aplicaciones de escritorio, sitios y aplicaciones web, servicios web para entornos compatibles con la plataforma .NET.
  - .NET incluye 2 frameworks, el clásico .NET Framework y el nuevo .NET Core. El primero se usa para creare aplicaciones Windows y el segundo permite modularidad y portabilidad a otras plataformas que no sean de Microsoft.
- Otros IDE's.
  - SharpDevelop.

Creación de una aplicación usando un IDE.

- **Solución.**
  - Contenedor para uno o más proyectos.
  - Una solución en un grupo de proyecto o sirve para agrupar proyectos con características comunes.

- **Proyecto.**
  - Conjunto de archivos que se compilan en un archivo ejecutable, biblioteca o sitio web.
  - Archivos que pueden ser de código fuente, iconos, imágenes, archivos de datos, etc.
  - Incluye también la configuración del compilador y otros archivos de configuración que podrían ser necesarios en diversos servicios o componentes con los que el programa se comunica.
  - Visual Studio utiliza MSBuild como herramienta y construcción de proyectos al compilar y generar ejecutables a partir del código fuente.
  - Un proyecto realizado con C# tendrá la extensión .csproj.
- **Ensamblado**.
  - Conjunto de tipos, módulos y recursos compilados para actuar como una unidad funcional.
  - Pueden ser archivos ejecutables (.exe) o bibliotecas de vínculos dinámicos (*.dll*).

Elementos del Lenguaje.

- **Comentarios.**

- **Uso:**
  - Documentar el código.
  - Añadir ideas, sugerencias, etc.
- **Sintaxis**:
  - Comentario de una línea:

```
// Texto del comentario.
```

  - Comentario de varias líneas:

```
/* Texto comentario línea 1
Texto comentario línea 2
Texto comentario línea N */
```

- Es **Case-Sensitive**, es decir, hace distinción ente mayúsculas y minúsculas, por ejemplo, las variables texto, Texto y TEXTO para C# son distintas.
- Toda instrucción o sentencia acaba siempre en punto y coma (;).
- Todo bloque de código va encerrado entre llaves {}.
- Para facilitar la comprensión del código y su legibilidad, es importante identarlo o sangrarlo.

ESTRUCTURA DE UN PROGRAMA EN C#

En un programa escrito en C se suelen incluir los siguientes apartados en el orden que se indica:

1. Inclusión de los **espacios de nombres** que incorpora C#.
2. Creación de un espacio de nombres por parte del desarrollador (opcional).
3. Declaración de una o varias clases:
   - Clases que pueden ser independientes o estar unas incluidas dentro de otras, es decir, ser internas o estar anidadas)
   - Instrucciones o sentencias que definen los datos y métodos.
   - Una de las clases o la única, si sólo hay una, será la clase principal e incluirá además de sus datos y métodos, el método Main ().

Espacio de nombres:

- Sirven para organizar las clases.
- Son una especie de contenedor para clases relacionadas.
- Puede ocurrir que distintos programadores creen funciones o clases que se llamen igual, y, si se mezclan fuentes de distintas procedencias, esto podría dar lugar a programas que se compilaran mal, o, tras ser compilados no funcionaran correctamente.
- De esta forma, los espacios de nombres permitan distinguir unos de otros,
- Sentencias para utilizar espacios de nombres:

using:

- Directiva que permite incluir espacios de nombres en una aplicación y las clases en ellos incluidas.
- Sintaxis:
  - using nombre del espacio de nombres
- Ejemplos:
  - using System;
  - using System.ComponentModel.Design;
  - using using System.Web;
- Si no se incluye la directiva using correspondiente, las clases a usar debe incluir el espacio de nombres delante.
- Ejemplo:
  - System.Console.WriteLine("¡Hola Mundo!");
- **system.**
  - Espacio de nombres que permite utilizar las funciones de la estructura básica de C# y de la plataforma .Net.

namespace:

- Permite al desarrollador de C# crear un espacio de nombres para sus clases.
- De esta forma puede tener mejor organizado el código.
- Sintaxis:
  - namespace nombre del espacio de nombres del programador {…}
- Ejemplo:
  - namespace prueba {...}

DECLARACIÓN DE UNA CLASE:

- El desarrollador puede crear sus propias clases.
- Sintaxis:

Modificadores de acceso class nombre de la clase.

{

Variables, constantes y otras estructuras de datos //atributos

Métodos (); //comportamiento o funcionalidad

Método Main (si es la clase principal)

}

Modificadores de acceso:

- Especifican accesibilidad a los datos desde dentro o fuera de la clase.
- Tipos:

    - **public**.
        - Acceso sin restricciones para los miembros de la clase desde otras clases.
    - **private**.
        - Miembros sólo accesibles desde dentro de la propia clase en la que se definen.
    - **internal.**
        - Sólo accesible los miembros de la clase dentro del mismo ensamblado.
    - **protected**.
        - Miembros accesibles sólo desde la clase que los incluye y desde las clases derivadas.
    - **static.**
        - No se pueden crear objetos y los métodos se pueden usar sin crear u objeto.
    - **abstract.**
        - Clase base para otras clases. No se pueden crear objetos con ellas y los métodos no están implementados, hay que hacerlo en las clases derivadas.
- Accesibilidad por defecto:
    - Los espacios de nombres son siempre public, modificador que no se incluye.
    - Clases son siempre internal.
    - Los miembros de una clase, atributos y métodos, son private.
    - Las estructuras o **struct** son private.
    - Los interfaces son public.

CREACIÓN DE UN OBJETO.

- Los objetos o instancias se crean a partir de una clase de la cual incluirán sus características**.**
- Se crean dentro del método Main() usando el operador new y el método constructor de la clase.
- Sintaxis:
    - Nombre de la clase nombre del objeto = new Método Constructor ();
- Llamar o utilizar un método o miembro de una clase se hace a través de un objeto utilizando el operador punto (.)
- Sintaxis:
    - Objeto.variable;
    - Objeto.Metodo();
- Ejemplo:
    - Metodo1 palabras = new Metodo1();
    - Circulo circulo1 = new Circulo ();
    - Metodo1 y Circulo son clases a partir de las cuales se crean los objetos palabras o circulo1.

MÉTODOS.

- No se pueden llamar igual que la clase, sólo los constructores.
- Sobrecarga de métodos:
    - Mecanismo que permite que puedan existir varios métodos con el mismo nombre, pero con distintos número o tipo de parámetros.
- Hay que crearlos dentro de la clase.
- Sintaxis:

1. **Métodos sin retorno de datos y sin parámetros**.

```
Modificador de acceso void Nombre ()
{
        Instrucciones;
}
```

2. **Métodos sin retorno de datos y con parámetros**.

```
Modificador de acceso void Nombre (tipo parametro1, tipo parámetro2,
...)
{
        Instrucciones;
}
```

3. **Métodos con retorno de datos y sin parámetros**.

```
Modificador de acceso tipo valor devuelto Nombre ()
{
        Instrucciones;
        return valor devuelto;
}
```

4. **Métodos con retorno de datos y con parámetros**.

```
Modificador de acceso tipo valor devuelto Nombre (tipo parametro1,
tipo parámetro2, …)
{
        Instrucciones;
        return valor devuelto;

}
```

MÉTODO MAIN().

- Método principal de una aplicación ya que es el punto de entrada a ésta, es decir, es el método a partir del cual se ejecuta inicialmente una aplicación.
- Sólo puede haber uno en un programa C#.
- Siempre se declara dentro de una clase.
- Es un método estático, es decir, no es necesario crear un objeto para poder utilizarlo.

- Sintaxis:
    - Hay varias sintaxis como, por ejemplo:
        - static void main () {}
        - static void main (String[] args){}
        - static int main (String[] args){}
        - static int main () {}

VARIABLES EN C#.

Para utilizar variables hay que seguir los siguientes pasos:

1. **Declaración**:

    - Las variables tienes un nombre único que las identifica y un tipo que indica que datos puede almacenar.
    - Nombre de una variable:

        - Puede incluir letras, números y algunos símbolos.
        - El primer carácter no puede ser un número.
        - C# hace distinción entre mayúsculas y minúsculas:
            - numero, Numero o NUMERO, son variables distintas.
        - No puede contener espacios en blanco, así que mejor usar un guion de subrayado o unir las palabras, como, por ejemplo:
            - sueldo_bruto
            - sueldoBruto
        - No puede haber dos variables con el mismo nombre.
        - No se pueden utilizar palabras reservadas de C#.
        - La mayor parte de caracteres especiales no sirven (,.*?).

    - Tipo de dato:
        - **Números enteros con signo**:
            - sbyte, short, int y long.
        - **Números enteros sin signo:**
            - byte, ushort, uint y ulong.
        - **Números reales o con coma flotante**:
            - float, double y decimal.
        - **Caracteres:**
            - **char**:
                - Un único carácter.
                - Se incluyen entre comillas simples.
            - **string:**
                - Cadenas de caracteres.
                - Se incluyen entre comillas dobles.
        - **Booleanos**:
            - **bool:**
                - Almacena uno de los siguientes valores booleanos: **true** o **false.**
    - Sintaxis:

    - tipo de dato nombre variable;
    - Pueden declarase a la vez varias variables del mismo tipo.
- Ejemplos:
    - int a;
    - string texto;
    - double sueldo;
    - int a, b, c;

2. **Asignación o inicialización de una variable**.

- Consiste en cargar un valor en una variable.
- Para ello se usa el operador de asignación el signo igual (=).
- Sintaxis:
    - nombre de la variable = valor;
- Ejemplos:
    - a = 7;
    - texto = “hola”;
    - sueldo = 1200.56;

3. **Creación y asignación de valores**.

- Se puede crear un variable y a la vez, asignarle un valor.
- Ejemplos:
    - int edad = 25;
    - char letra = ‘T’;

CONSTANTES EN C#.

- Se declaran y se les asigna el valor a la vez.
- Los tipos de datos son los mismos que los utilizados en las variables.
- El nombre de una constante sigue las mismas pautas que las utilizadas para crear una variable, si bien y aunque no es obligatorio, se suele poner su nombre en mayúsculas para distinguirlas.
- Sintaxis:
    - const tipo NOMBRE DE LA CONSTANTE = valor;
- Ejemplo:
    - const double PI = 3.1416;

Operadores.

- **Aritméticos**.
    - Unarios:
        - ++ Incremento, -- Decremento, + Signo positivo, - Signo negativo.
    - Binarios:
        - Multiplicar.
        - / Dividir.
        - + Sumar.
        - Restar.

        - % Módulo o resto de división.
- **Relaciones o de comparación**.
    - == Igual > Mayor <Menor >= Mayor o igual <= Menor o igual != Distinto
- **Lógicos.**
    - ! ----- Negación lógica.
    - && ---- Y lógica
    - || ----- O lógica.
- **Asignación**.
    - =
- **Asignación compuesta**.
    - += Suma y asignación. x+=y ➔ x=x + y
    - -= Resta y asignación. x-=y ➔ x=x - y
    - *= Multiplicación y asignación. x*=y ➔ x=x * y
    - /= División y asignación. x/=y ➔ x=x / y
    - %= Resto y asignación. x%=y ➔ x=x % y
- **Concatenación.**
    - Sirve para unir cadenas de caracteres o cadenas y variables, constantes, etc.
    - Se usa el signo +.

CLASE CONSOLE.

- Representa los flujos de entrada y salida para las aplicaciones de consola.
- Incluida en el espacio de nombres System.
- Deriva de la clase Object.
- **Métodos**.
    - **WriteLine ().**
        - Muestra por pantalla un texto o el contenido de una variable, constante, etc., y da un salto de línea.
        - Se puede incluir el operador de concatenación para combinar textos con variables.
        - Sintaxis:
            - Console. WriteLine(“Texto”);
            - Console. WriteLine(variable);
    - **Write ().**
        - Muestra por pantalla un texto o el contenido de una variable, constante, etc.
        - Se puede incluir el operador de concatenación para combinar textos con variables.
        - Sintaxis:
            - Console. Write(“Texto”);
            - Console. Write(variable);
    - **ReadLine ().**
        - Lee un dato desde el teclado y lo carga en una variable y se produce un salto de línea.
        - El dato entra en la variable al pulsar intro.

    - El dato entra en la variable como cadena de caracteres, por lo que si se necesita cargar en variables datos números hay que realizar una conversión de tipos.
    - Sintaxis:
        - variable = Console.ReadLine();
- **Read ().**
    - Lee un dato desde el teclado y lo carga en una variable.
    - El dato entra en la variable al pulsar intro.
    - El dato entra en la variable como cadena de caracteres, por lo que si se necesita cargar en variables datos números hay que realizar una conversión de tipos.
    - Sintaxis:
        - variable = Console.Read();
- **ReadKey ().**
    - Detiene la ejecución de un programa y la continua al pulsar una tecla.
    - Sintaxis:
        - Console.ReadKey();
- **Clear ().**
    - Limpia la pantalla.
    - Sintaxis:
        - Console.Clear();

CONVERSIÓN DE TIPOS.

- Para utilizar un dato es posible que antes haya que convertido al tipo adecuado.
- **Conversión implícita**:
    - Realizada automáticamente.
    - No se pierden datos.
    - Conversión de tipos de menor precisión a mayor precisión.
- **Conversión explícita**.
    - Se convierte un tipo numérico de mayor precisión o rango de valores a uno de menor rango o precisión.
    - Pueden perderse datos.
    - Se usa la expresión cast.
    - Sintaxis:
        - variable1 = (tipo de dato a convertir) variable2;
    - Ejemplo:
        - float a = 4.5f;
        - int b;
        - b = (int)a;
- **Método Convert**:
    - Clase que incluye multitud de métodos para convertir un tipo de datos en otro. (Ver documentación de C#).
    - Sintaxis:
        - Convert.ToTipo(dato a convertir);
- **Método Parse ().**
    - Método para convertir un tipo de datos en otro.

- Sintaxis:
  - Variable = tipo.Parse(dato a convertir);
- Ejemplo:
  - numero = int.Parse(teclado);

- **Conversión a cadenas de caracteres**.
  - Se usa el método ToString().
  - Sintaxis:
    - variable.ToString();
    - **variable**:
      - Nombre de la variable con el tipo de datos a convertir a una cadena.
  - Ejemplo:
    - int numero = 50;
    - string texto = numero.ToString();

INSTRUCCIONES CONDICIONALES O DE BIFURCACIÓN.

Condicional simple.

- Se realizan unas acciones o instrucciones si se cumple una condición, si no, no se realiza ninguna acción.
- Se usan operadores relacionales y lógicos para establecer las condiciones.
- Se pueden comparar variables con variables, variables con números literales, variables con texto, operaciones aritméticas con variables, ….
- Sintaxis:

```
If(condición)
  {
      Instrucciones;
  }
```

- Las expresiones condicionales que se establecen como condición pueden ser de 2 tipos:

- **Condición simple:**
  - Se establece una sola condición usando un operador relacional.
  - Ejemplos:
    - (a > b)
    - (a+b > c)
    - (a!=80)
    - (b==“hola”)
- **Condición compuesta:**
  - Se evalúan varias condiciones mediante operadores lógicos.
  - Y lógica (&&)
    - Para que se cumpla la condición global y se realicen las acciones deben cumplirse obligatoriamente todas las condiciones simples.
    - Ejemplo:
      - ((a>b) && (b>c))

- O lógica (||)
  - Para que se cumpla la condición global y se realicen las acciones deben cumplirse al menos una condición simple.
  - Ejemplo:
    - ((a>b) || (b>c))

Condicional completa.

- Se realizan unas acciones o instrucciones si se cumple una condición y si no, se realizan otras acciones.
- Se usan operadores relacionales y lógicos para establecer las condiciones.
- Se pueden comparar variables con variables, variables con números literales, variables con texto, operaciones aritméticas con variables, ….
- Se pueden usar condiciones simples o compuestas.
- Sintaxis:

```
If(condición)
    {
        Instrucciones si la condición se cumple;
    }
else
    {
        Instrucciones si la condición no se cumple;
    }
```

Condicional anidada.

- Se realizan unas acciones o instrucciones si se cumple una condición y si no, se evalúa una nueva condición y así sucesivamente si no se cumple la previa.
- Si no se cumple ninguna condición pueden, o no, realizarse otras acciones o instrucciones.
- Se usan operadores relacionales y lógicos para establecer las condiciones.
- Se pueden comparar variables con variables, variables con números literales, variables con texto, operaciones aritméticas con variables, ….
- Se pueden usar condiciones simples o compuestas.
- Sintaxis:

```
If(condición 1)
    {
        Instrucciones;
    }
    else if (condición 2)
    {
        Instrucciones;
    }
    else if (condición N)
    {
        Instrucciones;
```

```
}
else //Opcional
{
    Instrucciones;
}
```

Condicional Múltiple con switch…case.

- Evalúa varios valores para una variable y en caso de cumplirse se realizan unas instrucciones.
- Sentencias asociadas:
  - **Default:**
    - Opcional.
    - Si no se cumple ninguna condición se realizarían las instrucciones incluidas en default.
  - **Break**
    - Instrucción de salto para salir de switch.
- Sintaxis:

```
tipo variable =valor;
switch(variable)
        {
        Case valor 1:
                Instrucciones;
                Break;
        Case valor 2:
                Instrucciones;
                Break;
        Case valor N:
                Instrucciones;
                Break;
        Default: (opcional)
                Instrucciones;
                Break;
}
```

Agrupamiento de valores con switch...case

- Si varios valores se evalúan a la vez para realizar las mismas acciones o instrucciones sólo se incluyen las éstas en el último case.
- Ejemplo:

```
Case 80:
Case 90:
Case 100:
        Instrucciones comunes a los 3 valores anteriores;
        Break;
```

INSTRUCCIONES REPETITIVAS O BUCLES.

- **Bucle mientras - while.**
  - Realiza un número determinado de instrucciones mientras se cumpla una condición.
  - Cuando deje de cumplirse la condición se sale del bucle y se ejecutan las instrucciones que vengan a continuación.
  - Se puede combinar con instrucciones condicionales.
  - Sintaxis:

```
while(condición)
{
        Instrucciones;
}
```

- **Bucle haz…mientras – do…while.**
  - Realiza un número determinado de instrucciones mientras se cumpla una condición.
  - Se diferencia de while en que, si no se cumple la condición, al menos una vez se realizan las acciones.
  - Cuando deje de cumplirse la condición se sale del bucle y se ejecutan las instrucciones que vengan a continuación.
  - Se puede combinar con instrucciones condicionales.
  - Sintaxis:

```
do
{
        Instrucciones;

} while(condición);
```

- **Bucle for**.
  - Bucle que se ejecuta un número de veces.
  - Se usa un variable contadora con un valor inicial que se va incrementando o decrementando hasta un valor final que actúa como condición.
  - La variable contadora debe ser un número entero.
  - Sintaxis:

```
int variable contador;
for (valor inicial de la variable contador; condición; incremento o decremento)
{
        Instrucciones;
}
```

  - Instrucciones asociadas:
    - **Break**:
      - Permite salir de un bucle y no continuar con sus iteraciones si se cumple una condición.

- **Continue**:
  - Permite salir de la iteración actual en un bucle si se cumple una condición y no realizar las instrucciones que vengan a continuación.
  - El bucle continúa funcionando con las siguientes iteraciones.

APLICACIONES WINDOWS.

Concepto.

- Aplicaciones de ventana que presentan datos de una forma gráfica y más amigable que la ventana de línea de comandos.
- Para desarrollar este tipo de aplicaciones .NET proporciona la tecnología Windows Forms.

Características.

- La interfaz gráfica de una aplicación de ventana está compuesta de controles.
- Los controles forman parte de una jerarquía que permite a estos tener propiedades comunes.
- Visual Studio incorpora un diseñador visual para manipular controles de manera gráfica.
- El código que define la interfaz se genera de manera automáticamente.

Elementos necesarios para el diseño visual.

- Para agregar controles se necesitan 3 paneles:
  - El explorador de soluciones donde aparecerá, si se está en modo diseño visual, el cuadro de herramientas con el conjunto de controles disponibles.
  - La ventana de edición en la que podemos intercambiar las ventanas editor de código y el diseñador visual.
  - La ventana de propiedades, donde se ajustarán las características, formato, posición etc., de un control determinado.

Agregar controles a la ventana.

- **Métodos:**
  - Arrastrar y soltar el elemento a la posición deseada.
  - Clic en el elemento en el cuadro de herramientas y luego clic en la posición de la ventana donde se quiera insertar.

Operaciones con controles o elementos.

- **Selección de un control**.
  - Clic sobre el elemento a seleccionar.
- **Selección múltiple**.
  - Clic en un elemento.
  - Mayúsculas + clic en otros elementos a seleccionar.
  - Una vez seleccionados uno o varios elementos las operaciones que se realicen afectan a todos ellos.
- **Eliminar selección**.
  - Clic fuera del control o clic en otro control.

- **Cambio de posición.**
    - Arrastrar el control a la nueva posición.
- **Cambio de dimensiones.**
    - Arrastrar los tiradores que aparecen en las esquinas y laterales de un control cuando éste está seleccionado.
- **Eliminación de un control**.
    - **Varios métodos**:
        - Botón derecho del ratón sobre el control y seleccionar borrar.
        - Seleccionar el control y pulsar la tecla de suprimir.
        - Seleccionar el control y seleccionar el comando Editar / Borrar.
- **Copiar o mover un control.**
    - Seleccionar el control o controles.
    - Derecho / Copiar o Derecho / Cortar, o también, en la barra de menú seleccionar Editar / Copiar o Editar / Cortar.
- **Ajuste de propiedades.**
    - Seleccionar el control en la ventana de diseño o el en menú desplegable del panel de propiedades.
    - Seleccionar, ajustar o modificar las propiedades que interesen en el panel de propiedades.
    - El panel de propiedades también se muestra para un control pulsando en éste con el botón derecho del ratón y seleccionado la opción propiedades.

TIPOS DE CONTROLES.

Textbox.

- Cuadro de texto de una o varias líneas.
- Por efecto una línea.
- Si se quiere saltar de línea, se necesita que la propiedad a *AcceptsReturn* tenga el valor *true*.

RichTextbox.

- Permite cuadro de texto con formato similar la Notepad de Windows, es decir, permiten texto enriquecido.

Label.

- Control utilizado para representar cualquier tipo de texto, como etiquetas asociadas a otro control, títulos, etc.

Combobox.

- Listas o menús desplegables que contienen diferentes opciones paras elegir.
- También permiten escribir.
- Los elementos se incluyen con la propiedad *Items*.

RadioButton.

- Botones de opción que representan opciones excluyentes.
- Para que sean excluyentes hay que agruparlos.
- El grupo al que pertenece el botón se define mediante la propiedad G*roupname*.
- Para que varios botones formen parte del mismo grupo y sean mutuamente excluyentes entres si, deben tener el mismo nombre de grupo.

- En SharpDevelop, para que formen un grupo de opciones excluyentes, hay que incluir los botones en un contenedor GroupBox.
- Los botones de opción que estén fuera de un GroupBox, no funcionarán sincronizados con los que estén dentro.
- El estado de una opción se recupera o se ajusta inicialmente con la propiedad *isChecked*. (*Checked* en SharpDevelop).

Checkbox.

- Casillas de verificación.
- Permiten la selección de ninguna, una o varias opciones.
- El estado de una opción se recupera o se ajusta inicialmente con la propiedad *isChecked*. (*Checked* en SharpDevelop).

ASP.NET

- Active Server Pages - Paginas de Servidor Activas.
- Entorno de aplicaciones web de Microsoft que permite crear páginas web dinámicas.

Requisitos para desarrollar aplicaciones web con ASP.NET:

- Editor de código.
- .Net Framework o .Net Core
- Un servidor web como IIS (Internet Information Server).

Modelo Code-Behind

- Es aconsejable separar la página web del archivo con el código (C# o Visual Basic).
- Este código, denominado código subyacente, se incluye en un archivo independiente.
- Los archivos con código subyacente se guardan con las extensiones aspx.cs o aspx.vb, según el lenguaje de programación usado para crearlos.

Formularios Web o Web Forms:

- Son las páginas web de ASP.NET.
- Se guardan con la extensión .aspx
- Contenido:
  - XHTML Y HTML estático.
  - Etiquetas que definen los controles web que se ejecutan o procesan en el servidor.

- **Estructura de una página ASP.NET.**

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="Nombre clase.Default" %>
<!--<!DOCTYPE html >-->
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head runat="server">
        <title>Título</title>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    </head>
    <body>
        <form id="form1" runat="server">
            <div>
            </div>
        </form>
    </body>
</html>
```

- **Elementos de una de una página ASP.NET.**

Directiva @ Page

- Contiene información sobre la configuración y propiedades de la página.
- Sintaxis:
- <%@Page atributo1 =”valor” atributo2 =”valor”.....atributo3 =”valor”%>
- Atributos:
  - **Language.**
    - Especifica el lenguaje de programación con el que esta creado el código asociado al formulario web.
    - Sintaxis:
      - language = “Lenguaje de programación”
    - Ejemplo:
      - language = “C#”
  - **title.**
    - Permite incluir un título en el formulario web.
    - Sintaxis:
      - title = “Título”
    - Ejemplo:
      - title = “Página de Inicio”
  - **Codebehind**.
    - Indica el nombre del archivo con el código subyacente.
    - Sintaxis:
      - CodeBehind = “Nombre del archivo con el código asociado”
    - Ejemplo:
      - CodeBehind = “controles.aspx.cs”
  - **Inherits**.
    - Especifica el nombre de la clase dentro de archivo del código subyacente.
    - Sintaxis:
      - Inherits= “Espacio de nombres.Clase”
    - Ejemplo:
      - Inherits= “MiWeb.Controles”

- **AutoEventWireUp**.
  - Asocia automáticamente los eventos de la página con los métodos definidos en ella.
  - Sintaxis:
    - AutoEventWireUp = true o false;

Página maestra:

- Define la apariencia y el comportamiento estándar para todas las páginas del sitio.
- Lo que se incluya en ella aparece en todas.
- Utilización:
  - En la directiva @Page las páginas que incluyan el contenido común de la maestra deben incorporar el siguiente atributo:
    - MasterPageFile="~/Site.Master"

        - Añadir en el código la etiqueta <asp:Content…..></asp:Content>

Añadir controles.

- En una página ASP.NET se pueden incluir controles de formulario como botones de comando, botones de opción, cuadros de texto, listas desplegables, etc.
- Sintaxis:
    - <asp:Tipo de Control Atributo 1 =”Valor” Atributo 2 =”Valor” Atributo N =”Valor” ></asp:Tipo de Control>
- Ejemplo:
    - <asp:Button id="b1" runat="server" Text="Escribir Nombre" OnClick="b1_Click" />
- Atributos:
    - **id.**
        - Identificador único del control que se utiliza para acceder a las propiedades y métodos del objeto desde el código.
    - **runat**
        - Indica que un control debe ser procesado en el servidor.
        - Atributo que permite a ASP.NET identificar un control de servidor.
        - Sintaxis:
            - runat = ”server”
        - Ejemplo:
            - <form id="form1” runat="server">

SERVICIOS WEB.

Creación

- Visual Studio al crear un nuevo proyecto tenemos que seleccionar la opción ASP.NET Web Service.
- Pasos:
    - Crear una solución / proyecto nuevo.
    - Seleccionar aplicación web ASP.NET (.NET Framework).
    - Seleccionar proyecto vacío.
    - Una vez abierto el proyecto, con el botón derecho sobre su nombre en el explorador de proyectos, seleccionar Agregar / Nuevo elemento.
    - Seleccionar Servicio Web (ASMX).

Página asmx.

- Al crearse el proyecto, el elemento principal de un servicio web es la página asmx
- Ésta es similar una aspx, pero sin interfaz gráfica.

Directiva @ WebService

- Contiene información sobre la configuración y propiedades de la página.

- Sintaxis:
  - <%@WebService atributo1 =”valor” atributo2 =”valor”.....atributo3 =”valor”%>
- Atributos:
  - **Language.**
    - Especifica el lenguaje de programación con el que esta creado el código asociado al formulario web.
    - Sintaxis:
      - Language = “Lenguaje de programación”
    - Ejemplo:
      - Language = “C#”
  - **Codebehind**.
    - Indica el nombre del archivo con el código subyacente.
    - Sintaxis:
      - CodeBehind = “Ruta de acceso/Nombre del archivo con el código asociado”
    - Ejemplo:
      - CodeBehind = “App_Code/servicio.cs”
  - **Class.**
    - Indica el nombre de la clase dentro de archivo del código subyacente o asociado.
    - Sintaxis:
      - Class = “Espacio de nombres.Clase”
    - Ejemplo:
      - Class = “ejemplo.Soap”

Archivo de código asociado.

- Incluye una clase que hereda de Web Service.
- Incluye todos los métodos que el servicio web necesite.
- Cada método deberá estar declarado con el atributo [WebMethod].
- Sintaxis:

```
[WebMethod]
Método ()
{
        Instrucciones;
}
```

- Ejemplo:

```
[WebMethod]
public string Saludo()
  {
    return "Hola Mundo";
  }
```

Espacio de nombres.

- Al ejecutar el proyecto en Visual Studio se muestra una página web en la que se puede probar el funcionamiento el servicio web y consultar el documento WSDL.
- Los servicios web requieren espacios de nombres únicos para no confundirlos con los esquemas de otros.
- tempuri.org es el URL de espacio de nombres por defecto utilizado por los productos de desarrollo de Microsoft, como Visual Studio.
- Los espacios de nombres que tienen el formato de una URL web válida no deben tener contenido real en esa URL. Lo único que importa es que el espacio de nombres sea globalmente único.

Publicación de un servicio web a nivel local (localhost).

- Crear una carpeta en cualquier lugar del disco.
- Ubícate en el explorador de soluciones de Visual Studio
- Sobre el nombre del proyecto, pulsar con el botón derecho el ratón y seleccionar la opción *Publicar*.
- En la ventana *Destino*, seleccionar *Carpeta* y pulsar en *siguiente.*
- Pulsar en *Examinar* y seleccionar la carpeta creada en el primer punto.
- Pulsar en *Finalizar*.
- Copiar la carpeta con el sitio web publicado en la carpeta *wwwroot* del servidor *IIS.*
- Abrir el servidor *IIS* e iniciarlo *(si no lo está ya).*
- Sobre el nombre del servidor *IIS*, pulsar con el botón derecho el ratón y seleccionar la opción *Agregar Sitio Web*.
- Dar nombre al sitio web a agregar.
- Seleccionar la carpeta del sitio web a agregar dentro de la carpeta *wwwroot.*
- Si es necesario, especificar un numero de puerto.
- Al finalizar, pulsar en Aceptar.

Clase proxy.

Permiten comunicar una aplicación con un servicio web remoto.

- Es decir, sirve de interfaz entre el cliente y el servicio web.
- Funcionamiento:
    - El objeto proxy traduce las llamadas a los métodos en mensajes XML/SOAP que son enviados al servicio web.
    - Las respuestas generadas por el servicio web son convertidas por el objeto proxy al tipo correspondiente para ser manipuladas por la aplicación.
    - La clase proxy se genera partir del documento WSDL que describe el servicio web.
    - De esta forma, se pueden consumir servicios web de diferentes fabricantes que se están ejecutando en diferentes plataformas.

- Se pueden crear clases proxy para cualquier tipo de proyecto (Consola, Windows, ASP.NET), de modo que cualquier aplicación .NET puede ser cliente de un servicio web.

Crear una referencia en el servicio web. (Opcional).

- En el panel de proyectos de Visual Studio pulsar con el botón derecho sobre el nombre del proyecto.
- Seleccionar *Agregar / Referencias de Servicio.*
- En el cuadro que se abre, pulsar en *Detectar.*
- Seleccionar el servicio web.
- Aceptar.

Cliente.

- Aplicación que hará uso de los métodos del servicio web publicado en un servidor remoto.
- Aplicación que consumirá el servicio web
- Esto implica crear una aplicación, por ejemplo, una página web ASP.NET (Web Forms) con su correspondiente archivo de código asociado, que se vinculará con el servicio que va a consumir.
- **Contenido de la página ASP.NET**: (Extensión .aspx).
    - La interfaz de la aplicación con los contenidos que se requieran como formularios, tablas, etc.
- **Contenido del archivo asociado**. (Extensión .aspx.cs).
    - Hay que llamar a la nueva clase de proxy en el código, que se crea al vincular al cliente con el servicio web.
    - Para ello, se crea un objeto o instancia de la clase proxy que hace referencia al servicio web creado.
    - Luego se utiliza la referencia que se crea al vincular cliente con servicio web.
    - Por último, se invoca a sus métodos.
- **Creación de un objeto en C#:**
    - Dentro del objeto que responde a un evento y que provoca el uso del servicio (por ejemplo, un botón de comando), se crea un objeto de la clase proxy.
    - Sintaxis**:**
        - Nombre de la referencia web creada.Nombre de la clase que tiene el servicio web Nombre del Objeto = new Nombre de la referencia web creada.Nombre de la clase que tiene el servicio web( );
    - Ejemplo:
        - SW1.ServicioWeb1 miServicio = new SW1.ServicioWeb1 ();
        - SW1 sería la referencia creada.
- **Invocación o llamada a los métodos del servicio web.**
    - Hay que crear las variables necesarias para cargar los datos introducidos en el formulario ASP y también, para cargar datos o resultados

provenientes de las operaciones realizadas por los métodos del servicio web.

- Sintaxis:
  - Nombre de la variable = Nombre del objeto proxy.Metodo (Parámetros);
- Ejemplo:
  - int numero1= int.Parse(t1.Text);
  - int numero2 = int.Parse(t2.Text);
  - int suma = miServicio.Sumar(numero1,numero2);
  - t3.Text = suma.ToString();

- **Vincular el cliente con el servicio web**.
  - Pasos:
    - En el panel de proyectos de Visual Studio pulsar con el botón derecho sobre *Referencias*.
    - Seleccionar *Agregar Referencias de Servicio.*
    - En el cuadro que se abre, pulsar en el botón *Avanzadas.*
    - En el cuadro siguiente, pulsar en *Agregar Referencia Web*.
    - En *Dirección URL*, especificar la URL local o remota del servicio web. (Si no se conoce, se puede consultar el documento WSDL del servicio Web).
    - Con esto, aparecerán los servicios web encontrados en la URL.
    - Dar nombre a la referencia web.
    - Pulsar en *Agregar referencia*.
    - La referencia se creará y aparecerá en el panel de proyectos.
- **Referencia correcta al localhost**.
  - Para establecer la referencia correctamente, hay que asegurarse de que:
    - El servicio web esté guardado en wwwroot del IIS.
    - El sitio web del servicio ha sido agregado al servidor.
    - El sitio web del servicio está iniciado.
  - Dirección a colocar en la referencia al servicio:
    - http://localhost/ruta de acceso/nombre de la página con el servicio web.asmx.
      - URL al servicio Web.
    - http://localhost/ruta de acceso/nombre de la página con el servicio web.asmx?WSDL.
      - URL al documento WSDL con la descripción del servicio.
  - Ejemplos:
    - http://localhost/website1/WebService.asmx
    - http://localhost/website1/WebService.asmx?WSDL

- **Publicación de un cliente para consumir un servicio web a nivel local (localhost).**
  - Crear una carpeta en cualquier lugar del disco.
  - Ubícate en el explorador de soluciones de Visual Studio
  - Sobre el nombre del proyecto, pulsar con el botón derecho el ratón y seleccionar la opción *Publicar*.
  - En la ventana *Destino*, seleccionar *Carpeta* y pulsar en *siguiente.*
  - Pulsar en *Examinar* y seleccionar la carpeta creada en el primer punto.

- Pulsar en *Finalizar*.
- Copiar la carpeta con el sitio web publicado en la carpeta *wwwroot* del servidor *IIS.*
- Abrir el servidor *IIS* e iniciarlo *(si no lo está ya).*
- Sobre el nombre del servidor *IIS*, pulsar con el botón derecho el ratón y seleccionar la opción *Agregar Sitio Web*.
- Dar nombre al sitio web a agregar.
- Seleccionar la carpeta del sitio web a agregar dentro de la carpeta *wwwroot.*
- Especificar un puerto distinto al que esté utilizando el servicio web.
- Al finalizar, pulsar en Aceptar.
- Iniciar tanto el servicio como el cliente web en el servidor IIS, si no lo están ya.
- Abrir el cliente web y utilizar sus funciones, para recibir la respuesta del servicio.

PHP Y SOAP.

- Los Servicios Web son funciones o clases remotas usadas por un programa en local sin importar su lenguaje de escritura.
- Los Servicios Web se suelen ejecutar en potentes servidores y usar (consumir) en dispositivos de menor potencia como móviles o PC's. Es una forma de reducir el trabajo de una aplicación.
- A partir de la versión 5, PHP incluye las clases SoapServer y SoapClient para crear y consumir servicios web.
- Además, existen librerías de terceros que pueden usarse para el mismo fin como NuSoap, que se descarga y descomprime en el directorio donde está la aplicación que será el Servicio Web.

SERVICIO PHP.

- Para crear un servicio web de tipo SOAP con PHP se usa la clase SoapServer.

CLIENTE PHP.

- Para crear un cliente SOAP con PHP se usa la clase SoapClient.
- Sintaxis:
  - $nombre del cliente = new SoapClient("archivo WSDL" [, array(opciones)]);
  - Argumentos.
    - **archivo WSDL.**
      - URI al archivo de descripción del servicio WSDL.
      - Si no existe el archivo de descripción del servicio, es decir, si se funciona en modo no-WSDL, se incluye el valor *null*.

        - Argumento obligatorio.
    - Opciones.
        - Se incluyen dentro de un array asociativo y son opcionales.
        - Algunas opciones:
            - **location**.
                - URL a la página web del servicio.
                - Es decir, URL del servidor SOAP al que enviar la petición por parte del cliente.
                - Obligatorio en modo no-WSDL.
                - Opcional en modo WSDL.
            - **uri.**
                - Espacio de nombres destino del servicio SOAP.
                - Obligatorio en modo no-WSDL.
                - Opcional en modo WSDL.
            - **trace**.
                - Activa el seguimiento de la petición para que los fallos puedan ser trazados.
                - Valor booleano (true/false).
                - Por defecto, *false*.
            - **exceptions**.
                - Establece si se lanzan excepciones del tipo SoapFault al producirse un error.
                - Valor booleano (true/false).
            - ***encoding.***
                - *Establece* la codificación interna de caracteres.
                - *Por defecto,* la codificación de las peticiones SOAP son siempre en utf-8.
- Ejemplos:
    - $archivo_wsdl = “http://www.xmethods.net/wsdl/query.wsdl”;
    - $cliente1 = new SOAPClient ($archivo_wsdl); // Sintaxis básica.
    - $cliente2 = new SOAPClient (“http://localhost/servicio.php?WSDL”); // Sintaxis básica.
    - $wsdl = “http://localhost/servicio.asmx?WSDL”;
    - $url_servicio=“http://localhost/servicio.asmx”;
    - $cliente3 = new SOAPClient ($wsdl, array(”location”=>$url_servicio, “trace” => true, “exceptions”=> false)); // Sintaxis con algunas opciones.
- **Creación de un cliente PHP para servicio ASP.NET.**
    - El servicio web debe estar iniciado en el servidor**.**

- Crear el objeto SOAPClient.
- (Opcional) Crear las variables que recibirán los datos cargados desde un formulario usando $_GET[], $_POST[] o $_REQUEST[].
- Crear un array asociativo con los valores a pasar a los métodos o funciones del servicio web.
    - Sintaxis:
        - $nombre del array = array ("nombre del parámetro 1" => valor 1, "nombre del parámetro 2" => valor 2, …, "nombre del parámetro N" => valor N);
    - Los nombres de los parámetros tienen que coincidir con los nombres usados como parámetros en los métodos o funciones del servicio web.
    - Los valores tienen que ajustarse a los tipos de datos definidos en los métodos.
    - Los valores pueden ser literales o variables que previamente han recibido dichos valores a través de formularios (ver 2º punto).
    - Si el método a llamar no tiene parámetros, no es necesario crear este array.
- Invocar o llamar al método del servicio web que se quiera usar.
    - Esta llamada devolverá (return) un resultado como un objeto de la clase stdClass, por lo que hay que crear un objeto para cargar éste.
    - **stdClass.**
        - Clase predefinida en PHP que no tiene ni propiedades, ni métodos, ni padre, es decir, no hereda de ninguna otra clase.
        - Es una clase vacía que se puede usar cuando se necesite crear un objeto genérico al que luego se le pueden añadir propiedades.
    - Sintaxis:
        - $nombre del objeto = nombre del objeto SoapClient->método (array con los parámetros);
        - El objeto creado contendrá un par propiedad-valor o clave-valor.
        - La propiedad o clave, que servirá para acceder al valor, tendrá el siguiente formato:
            - NombreDelMetodoResult.
- Visualizar el resultado.
    - Para acceder al dato devuelto, se utiliza su propiedad o clave a través de objeto que contiene el resultado.
    - Para ver el resultado se puede usar cualquier función de salida

de datos como echo(), printr(), var_dump(), etc.

- También se puede visualizar el resultado usando un bucle foreach.
- Sintaxis:
  - $nombre del objeto->NombreDelMetodoResult;

- Ejemplo:
  - $archivo_WSDL= "http://localhost/ventas.asmx?wsdl"; //URL al WSDL.
  - $subtotal_SOAP = new SoapClient($archivo_WSDL); // Creación objeto cliente.
  - $cantidad = $_REQUEST["cantidad]; // Recepción de valor desde un formulario y carga en variable.
  - $precio = $_REQUEST["precio"]; // Recepción de valor desde un formulario y carga en variable.
  - $parametros_subtotal = array("valor1"=>$cantidad,"n2" => $precio); // creación de un array con los valores a utilizar como parámetros en el método del servicio web a usar.
  - $resultado_subtotal = $subtotal_SOAP->CalcularSubtotal ($parametros_subtotal); // Llamada a través del objeto cliente al método del servicio web y carga del resultado devuelto en una variable, que se convierte en un objeto.
  - echo " Resultado suma: " .$resultado_subtotal->CalcularSubtotalResult." "; // Visualización del resultado.