Lenguaje C#

Introducción

C# es el último en una línea de evolución de los lenguajes derivados de C que incluye C++ y Java.
Ha sido utilizado por Microsoft para desarrollar la mayoría de código de .net. Es un lenguaje orientado a objetos simple y potente.

Case-sensitive

El lenguaje C#, al igual que sucede con Java, distingue entre mayúsculas yminúsculas, es decir, es **case-sensitive.** Por tanto no es la misma variable nombre que la variable Nombre.

Bloques

Al igual que Java, C# utiliza las llaves {} para determinar los bloques dentro de un programa. Todo lo que se encuentra entre estas dos llaves se considera un bloque. Estos bloques definen el **ámbito de las variables**, es decir, la existencia de las mismas. Cuando finaliza el bloque ésta es eliminada y la memoria que ocupaba es liberada por el recolector de basura (garbage collector)

Los bloques pueden anidarse, suele ser de lo más común y se utiliza la sangría para que el contenido de los mismos sea fácilmente reconocible.

Comentarios

Existen dos formas de realizar comentarios en programas escritos en C# . Los comentarios de una única línea, a los que se les antepone dos barras inclinadas (//). Ejemplo:

```
//Este es un comentario de una línea.
```

Los comentarios de varias líneas se encontrarán entre los símbolos /* y */. Ejemplo:

```
/* Este es un comentario
de
varias líneas */
```

Identificadores

Son los que dan nombre a variables, constantes y métodos. Deben seguir las siguientes normas:

- Constar de caracteres alfanuméricos y _
- C# es **Sensible** a mayúsculas y minúsculas.
- Han de comenzar con letra o _
- No pueden ser una palabra reservada…

abstract, as, base, bool, break, byte, case, catch, char, checked, class, const, continue, decimal, default, delegate, do, double, else, enum, event, explicit, extern, false, finally, fixed, float, for, foreach, goto, if, implicit, in, int, interface, internal, lock, is, long, namespace, new, null, object, operator, out, override, params, private, protected, public, readonly, ref, return, sbyte, sealed, short, sizeof, stackalloc, static, string, struct, switch, this, throw, true, try, typeof, uint, ulong, unchecked, unsafe, ushort, using, virtual, void, while.

...si no se usa el carácter '**@**' como prefijo. Ejemplo:

int @as=0;

Declaración de variables y Tipos de Datos

Todos los lenguajes de.NET comparten un sistema común de tipos (CTS). Se admite, tanto tipos de tipo valor como tipos de referencia:

Todos los tipos derivan de la clase **System.Object.** La relación de todos ellos es la siguiente:

Tipo	Descripción	Bits	Rango de valores	Alias
SByte	*Bytes* con signo	8	[-128, 127]	sbyte
Byte	*Bytes* sin signo	8	[0 , 255]	byte
Int16	Enteros cortos con signo	16	[-32.768, 32.767]	short
UInt16	Enteros cortos sin signo	16	[0, 65.535]	ushort
Int32	Enteros normales	32	[-2.147.483.648, 2.147.483.647]	int
UInt32	Enteros normales sin signo	32	[0, 4.294.967.295]	uint
Int64	Enteros largos	64	[-9.223.372.036.854.775.808, 9.223.372.036.854.775.807]	long
UInt64	Enteros largos sin signo	64	[0-18.446.744.073.709.551.615]	ulong
Single	Reales con 7 dígitos de precisión	32	$[1{,}5\times10^{-45} - 3{,}4\times10^{38}]$	float
Double	Reales de 15-16 dígitos de precisión	64	$[5{,}0\times10^{-324} - 1{,}7\times10^{308}]$	double
Decimal	Reales de 28-29 dígitos de precisión	128	$[1{,}0\times10^{-28} - 7{,}9\times10^{28}]$	decimal
Boolean	Valores lógicos	32	true, false	bool
Char	Caracteres Unicode	16	['\u0000', '\uFFFF']	char
String	Cadenas de caracteres	Variable	El permitido por la memoria	string
Object	Cualquier objeto	Variable	Cualquier objeto	object

Una variable representa la localización en memoria donde se guarda una instancia de un tipo.

Para **declarar** una variable, debemos indicar el **tipo de la variable** (tipo simple o clase) y a continuación pondremos el **nombre de la variable**. Si lavariable posee algún modificador de acceso o de otro tipo, este modificador aparecerá antes del tipo de la variable.

Debemos recordar que los nombres de las variables deberán ser escritas con la primera letra de cada una de las palabras que la forman en mayúscula.

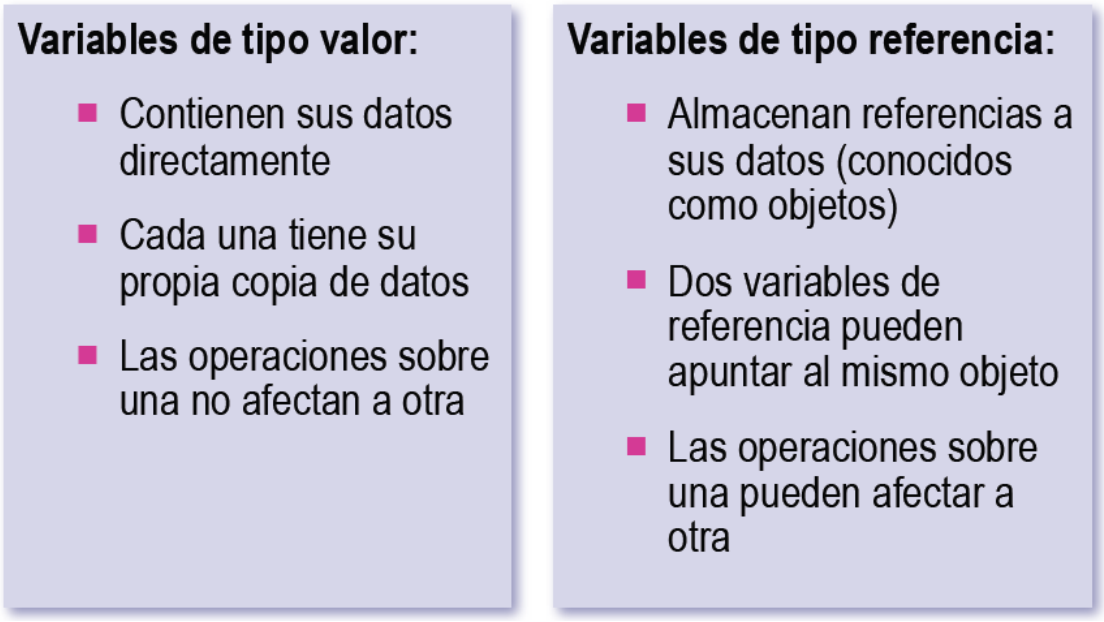
Sintaxis:

```
[ámbito]<tipo><variable>;
[ámbito] <tipo><variable> = <valor>;
```

Ejemplos:

```
private int OperandoSuma;
public DateTime FechaNacimiento;
string Cadena;
bool Encontrado;
```

Debemos distinguir entre variables de tipo valor y variables de tipo referencia.



Inicialización de variables en la declaración:

```
bool bln = true;
byte byt1 = 22;
char ch1='x', ch2='\u0061'; // Unicode para 'a'
decimal dec1 = 1.23M;
double dbl1=1.23, dbl2=1.23D;
short sh = 22;
int i = 22;
long lng1=22, lng2=22L; // 'L' long
```

```
sbyte sb=22;
float f=1.23F;
ushort us1=22;
uint ui1=22, ui2=22U; // 'U' unsigned
ulong ul1=22, ul2=22U, ul3=22L, ul4=2UL;
```

Los valores de referencia son creados con la palabrareservada **new**:

```
char[] discos = new char[10];
```

Una variable de tipo **String** se puede inicializar directamente:

```
string s = "Hola";
```

C# soporta secuencias de escape como en C:

```
string s1 = "Hola\n"; // salto de línea
string s2 = "Hola\tque\ttal"; // tabulador
```

Secuencias de escape	
Alerta (timbre)	\a
Retroceso	\b
Avance de página	\f
Nueva línea	\n
Retorno de carro	\r
Tabulador horizontal	\t
Tabulador vertical	\v
Nulo	\0
Comilla simple	\'
Comilla doble	\"
Barra inversa	\\

Como las sentencias de escape comienzan con '\', para escribir este carácter hay que doblarlo, o usar '@':

```
string ruta = "c:\\Archivos";
string rutawindows = @"C:\WINDOWS";
```

Àmbito y Visibilidad de las Variables

Ámbito: lugar donde se declara una variable. Determina su accesibilidad.
La Visibilidadde una variable fuera de su ámbito se puede modificar anteponiendo **public** o **private** en su declaración.

static: Variable estática**,** existe durante toda la ejecución del programa. Sólo existe una sola copia de la misma sin necesidad de crear objeto alguno. Hay que referirse a ella usando el nombre completo del tipo al que pertenece.
No afecta a su visibilidad

```
public static int Var3=30; // pública
// estàtica
static int Var4=40; //privada y estàtica
```

Constantes

Se definen usando el modificador const.

Las constantes son implícitamente static y no pueden cambiar nunca de valor.

```
private const int min = 1;
private const int max = 100;
public const int rango = max – min;
```

Conversión entre tipos de datos


```
int intValor = 123;
long longValor = intValor; // implícita de int a long
intvalor = (int) longValor; // explícita de long a int con cast
int x = 123456;
long y = x; // implícita
short z = (short)x; // explícita
double d = 1.2345678901234;
float f = (float)d; // explícita
long l = (long)d; // explícita

//uso de la clase Convert:

int c=154;
lblresultado.Text=Convert.ToString(c);
```

Tabla de conversión implícita segura

Tipo	Se convierte de forma segura a:
Byte	short, ushort, int, uint, long, ulong, float, double, decimal
Sbyte	short, int, long, float, double, decimal
Short	int, long, float, double, decimal
Ushort	int, uint, long, ulong, float, double, decimal
Int	long, float, double, decimal
Uint	long, ulong, float, double, decimal
Long	float, double, decimal
Ulong	float, double, decimal
Float	double
Char	ushort, int, uint, long, ulong, float, double, decimal

Manipulación de cadenas

Los tipos **char** y **string** permiten manipular caracteres.
Una variable **char** puede contener cualquier carácter Unicode

Manipulacion de caracteres: char.IsDigit(…);
char.IsLetter(…); char.IsPunctuation(…);
char.ToUpper(…); char.ToLower(…),
char.ToString(…);..., etc.

Una variable tipo **string** es una referencia al lugar donde se guarda la cadena.
Cada vez que se modifica el valor de la cadena se asigna un nuevo bloque de memoria y se libera el anterior.

Concatenación: Nuevo operador **+** (no existe en C, C++) o se puede usar también el método: **String.Concat()**

```
string a, b;
a="Diseño y Programación Web";
b="con C#";
string.Concat(a, b));
```

Operadores

Los operadores nos van a ayudar a realizar operaciones sobre uno o más operandos. Así pues existen operadores unarios (los que actúan sobre un único operando), binarios (los más habituales que actúan sobre dos operandos).

Categoría del Operador	Operadores
Unarios	+,-,!,~,++x,--x, (T)x
Aritméticos	*,/,%,+,-
Desplazamientos	<<,>>
Relacionales	==,<,<=,>,>=, is
Lógicos	&,&&,|,||,^,?:
Asignación	=,*=,/=,%=,+=,-=,<<=,>>=,&=,^=,|=

A parte de los operadores nombrados existen algunos otros explicados aquí:

- El operador **[]** se utiliza para hacer referencia a los elementos de un **array**. Todos los arrays en .NET **comienzan en 0**.
- Los operadores (++ y --) son operadores de **autoincremento** y **autodecremento**.
- **new** es el operador usado para invocar **métodos constructores** a partir de definiciones de clases.
- **typeof** es el operador utilizado mediante la librería Reflection para obtener **información acerca de un tipo**.
- **sizeof** se utiliza para determinar el **tamaño de un tipo estándar** expresado en bytes. Solo puede ser utilizado con los llamados value types (tipos por valor), o miembros de clases, pero no con las propias clases. Además, sólo pueden usarse dentro de bloques unsafe (código C++ embebido en un bloque unsafe de C#).
- **checked** y **unchecked** se utilizan para la **gestión de errores**, dentro de estructuras try-catchfinally.
- Los **operadores de asignación compuesta** se utilizan para simplificar expresiones en las que un operando aparece en los dos miembros de la asignación. Por ejemplo, podemos simplificar un contador expresándolo como **x += 1,** en lugar de **x = x + 1.**

Sentencias condicionales

En C# disponemos de la sentencia condicional **if ...else...** Su sintaxis es la siguiente:

```
if(condición){
.....
}else{
.....
}
```

Por ejemplo:

```
if (a==1){
      lblresultado.text="El contenido es correcto";
}else{
      lblresultado.text="El contenido no es el deseado";
}
```

También tenemos en C# como sentencia condicional la sentencia **switch...case...** Su sintaxis es la siguiente:

```
switch(expresión){
case expresión_constante:
      ........
      break;
[default:
      ...
      break;]
}
```

En cada condición deberemos indicar la sentencia break sino queremos que se sigan evaluando lassiguientes condiciones. La rama default no es obligatoria, pero se ejecutará siempre que no se cumplan ninguna de las condiciones anteriores.

Ejemplo:

```
switch (cadena){
case "Uno":
      lblresultado.text="1";
      break;
case "Dos":
      lblresultado.text="2";
      break;
case "Tres":
      lblresultado.text="3";
      break;
default:
      lblresultado.text="Otro número";
      break;
}
```

Sentencias Repetitivas

Una sentencia repetitiva que podemos encontrar en C# es la sentencia while.

Mediante esta sentencia podemos ejecutar un conjunto de instrucciones mientras se cumpla una determinada condición. Su sintaxis es:

```
      REPETITIVA 0-N

while(expresión) {
      ...
      sentencias;
      ...
}
```

```
      REPETITIVA 1-N

do {
      ...
      sentencias;
      ...
} while(expresión)
```

Ejemplo:

```
int x=0;
while (x<=10){
      cbonúmeros.Items.add(x);
      x++;
}
```

En C# existen dos tipos de bucles for, el **bucle for** al estilo de **Java**, cuya sintaxis es la siguiente y un bucle **foreach** similar al for each de **Visual Basic**.

```
for(inicialización;expresión;iteradores){
      ...
      sentencias;
      ...
}
```

- En **inicialización** se inicializa la variable contador que vamos a utilizar en el bucle.
- En **Expresión** se define la condición que se debe cumplir para que no finalice el bucle.
- En **Iteradores** se colocará una sentencia o conjunto de ellas que incrementará o decrementará el contador del bucle.

Ejemplo:

```
for (int i=0; i<=12; i++){
      cbomeses.Items.Add(i);
}
```

La sintaxis general del **bucle foreach** es:

```
foreach (tipo identificador in expresión){
      ...
      sentencias;
      ...
}
```

La expresión en el bucle **foreach** será el nombre de un array o de una colección. La sentencia **foreach** seutiliza para recorrer los elementos de una colección.

```
String [] meses =
{"Enero","Febrero","Marzo","Abril","Mayo","Junio","Julio","Agosto","Se
ptiembre","Octubre","Noviembre","Diciembre"};


foreach(String mes in meses){
      cbomeses.Items.Add(mes);
}
```

Manejadores de eventos

Estos son métodos especiales que se van a encargar del tratamiento de eventos en el cliente, pero queejecutarán código de servidor. Estos métodos los veremos en detalle cuando veamos más adelante los**Web Forms** y los controles de servidor.
Los métodos para el tratamiento de eventos poseen dos parámetros, el primero de ellos es el controlque generó el evento (la fuente u origen del evento), y el segundo parámetro serán los argumentosadicionales que necesite el método, normalmente estos argumentos dependen del tipo de evento atratar.

```
void NombreMetodo (Object origen, EventArgs argumentos){
      ...
      sentencias;
      ...
}
```

A continuación se ofrece un código en el que se puede ver un método que va atratar el evento de pulsación de un botón dentro de un Web Form. Para que el código sea máscompleto se ha incluido también el Web Form y el botón que genera el evento al ser pulsado, en estecódigo se adelanta lo que podremos ver a partir del siguiente capítulo, los Web Forms y los controlesde servidor de ASP .NET.

```
<html>
<body>
<script language="C#" runat="server">
      void Pulsado(Object origen, EventArgs argumentos){
            etiqueta.Text ="El origen del evento es: "+
            origen.ToString();
      }
</script>
<form id="formulario" method="post" runat="server">
      <asp:button id="boton" text="Pulsar" onclick="Pulsado"
      runat="Server"></asp:button>
      <asp:label id="etiqueta" runat="server"></asp:label>
</form>
</body>
</html>
```

Conversión de tipos

El lenguaje C# es un lenguaje fuertemente tipado, es decir, es bastante estricto en lo que a conversión y compatibilidad de tipos de datos se refiere.
C# al igual que C y Java admite el moldeado de tipo (casting) o conversión de tipos, lo que nospermite convertir un tipo de dato en otro. Lo que se suele utilizar principalmente para

cambiar el tipo de dato devuelto por una función para que coincida con el tipo de dato esperado por otra a la que se lo vamos a pasar como parámetro.
Para realizar el moldeado de tipo (casting), se coloca el tipo desea entre paréntesis a la izquierda del valor que vamos a convertir.
En el Código fuente siguiente se puede comprobar cómo se utiliza el mecanismo de casting, para convertir un objeto de la clase int en un objeto de la clase long.

```
int num1=25;
long num2=(int)num1;
```

Además del mecanismo de casting, el lenguaje C# ofrece en las distintas clases una serie de métodos que permiten la conversión de tipos. Así por ejemplo si queremos convertir un objeto entero a una cadena de caracteres, utilizaremos el método ToString() de la clase de tipo simple int. También podemos tener el caso opuesto, en el que queremos convertir una cadena a un entero, para ello se utiliza el método Parse(). En el Código siguiente se pueden ver estas situaciones.

```
int i=3;
String s=i.ToString();
int numero=int.Parse("10");
```

Tipos definidos por el usuario: Enumeraciones

Las enumeraciones se crean cuando se necesita que una variable tome valores de una lista limitada no predefinida.

Tienen por tipo base números enteros.
Por defecto empieza la numeración por 0.

```
enum Color { Rojo=1, Verde=2, Azul=3 }
```

Uso de una enumeración:

```
Color colorPaleta = Color.Rojo;
```

o

```
colorPaleta = (Color)1; // Tipo int a tipo Color
```

Visualización de una variable de enumeración:

```
txtColor.Text=colorPaleta.ToString());
```

```
// Muestra Rojo
```

Usando enumeraciones el código queda más claro y fácilde mantener.

Tipos definidos por el usuario: Estructuras

Definición:

```
struct Empleado {
public string nombre;
public int edad;
public int Sueldo = 1200; // error, no permitido!
public Empleado(string nom, int edad … )
}
```

Uso:

```
Empleado empresaEmpleado, otroEmpleado;
empresaEmpleado.Nombre = "Juan";
empresaEmpleado.edad = 23;
otroEmpleado = empresaEmpleado; //asignación directa
```

Todos los miembros se inician por defecto a **null**.

El método booleano **Equals()** permite comparar dos estructuras.

```
otroEmpleado.Equals(empresaEmpleado)
```

Arrays

Los arrays son tipos de referencia derivados de System.Array.
Sus índices comienzan en 0.

Ejemplo: `string[] a; // array de cadenas`

El tipo de datos array viene dado por string[]
El nombre del array es una referencia al array

Para crear espacio para los elementos usar:

```
string[] a = new string[100];
```

Los arrays se pueden inicializar directamente:

```
string[] animales = {"gato", "perro", "caballo"};
int[] a2 = {1, 2, 3};
```

Arrays multidimensionales:

El rango de los elementos del array es dinámico.
Si se crea un nuevo array sobre el mismo se libera la memoria que ocupaba y se pierde toda la información que contenia.

```
string[,] ar = {{"perro","conejo"}, {"gato","caballo"}};
double[,] tempAnual = new double[12,31];
```

Información sobre un array:

- Dimensiones: .Rank
- Número total de elementos del array: .Length
- Número de elementos de la dimensión **d**: .GetLength(d)
- Índice superior e inferior: .GetLowerBound(d);.GetUpperBound(d);
- (d=dimensión, desde 0 hasta Rank-1)
- Saber si es un array: if (a is Array)

Recorrido de los elementos de un array sin conocer sus índices

```
foreach (string a in animales)
Console.WriteLine(a);
```

Operaciones de la clase Array: Copy( ); Sort( ); ...

FUNCIONES EN C# .NET

1.- FUNCIONES CON NÚMEROS

A parte de las operaciones anteriormente vistas (+ , -, *, /, \, Mod), Visual Basic es capaz de realizar operaciones matemáticas más complejas gracias a la clase **System.Math**.

FUNCIONES GENERALES

Estas funciones nos permiten realizar algunas operaciones matemáticas casi complejas.

Función	Descripción
Abs	Devuelve el valor absoluto del número dado
Sign	Devuelve el signo de un número
Sqrt	Halla la raíz cuadrada del número dado
Exp	Calcula el número e elevado al exponente indicado
Log	Halla el logaritmo natural del número dado
Log10	Halla el logaritmo en base 10 del número dado
Max	Devuelve el mayor de dos números dados
Min	Devuelve el menor de dos números dados
Pow	Devuelve un número especificado elevado a la potencia especificada.
Floor	Devuelve el número entero más grande menor o igual que el número especificado.
Round	Devuelve el número más cercano al valor especificado.
Ceiling	Devuelve el número entero más pequeño mayor o igual que el número especificado.

La función *Abs* toma como parámetro cualquier número, positivo o negativo, y devuelve su valor absoluto, lo que equivale a eliminar el signo de números negativos.

Sign devolverá 1 si el número que se pasa como parámetro es positivo, -1 si es negativo y 0 si el número es 0, ya que éste no se considera ni negativo ni positivo.

FUNCIONES TRIGONOMÉTRICAS

Visual Basic no es capaz de trabajar con grados en temas relacionados con ángulos, Visual Basic trabaja con radianes, por lo que será necesaria la conversión cada vez que la unidad no sea ésta.

Función	Descripción
Sin	Devuelve el seno del ángulo especificado.
ASin	Devuelve el ángulo cuyo seno es el número especificado.
Cos	Devuelve el coseno del ángulo especificado.
ACos	Devuelve el ángulo cuyo coseno es el número especificado.
Tan	Devuelve la tangente del ángulo especificado.
Atan	Devuelve el ángulo cuya tangente corresponde al número especificado.

Si necesita convertir una medida de grados a radianes, antes de pasarla como parámetro a *Sin, Cos* o *Tan,* multiplique el número por PI y divida entre 180. La conversión en sentido inverso, para convertir los radianes devueltos por *Atn* en grados, la conseguirá multiplicando por 180 y dividiendo por PI.

NÚMEROS ALEATORIOS

En .NET existe la clase **Random** para generar números aleatorios.

La clase **Random** representa un generador de números pseudoaleatorios, un dispositivo que genera una secuencia de números que cumplen determinados requisitos estadísticos de aleatoriedad.

Los números pseudoaleatorios se eligen con la misma probabilidad en un conjunto finito de números. Los números elegidos no son completamente aleatorios porque se seleccionan mediante un algoritmo matemático concreto, pero su aleatoriedad es suficiente para fines prácticos.

La generación de números aleatorios comienza por un valor de inicialización. Si se utiliza la misma inicialización repetidas veces, se genera la misma serie de números. Una manera de

generar secuencias diferentes es hacer que el valor de inicialización dependa del tiempo y, por tanto, se genere una serie diferente con cada nueva instancia de **Random**.

Los métodos más usados de esta clase son:

Next: Devuelve un número aleatorio.

Ejemplo en C#:

```
Random R= new Random();
txtetiqueta.Text=R.Next (0,256).ToString();
```

Ejemplo en Visual Basic .NET

```
Dim R as New Random
txtetiqueta.text=R.Next (0,256)
```

Estas secuencias generarán un número aleatorio entre 0 y 255. El primer parámetro del método next es el menor número que puede generar y el segundo parámetro es siempre el valor mayor que el último que puede generar.

NextBytes:Rellena con números aleatorios los elementos de una matriz de bytes

Ejemplo en C#

```
Random R=new Random();
Byte b(10)=new Byte;
rnd.NextBytes(b)
Console.WriteLine("The Random bytes are: ")
Dim i As Integer
For i = 0 To 9
   Response.Write(i)
   Response.Write(":")
   Response.WriteLine(b(i))
next
```

Ejemplo en Visual Basic .NET

```
Dim R As New Random()
Dim b(10) As Byte
rnd.NextBytes(b)
Console.WriteLine("The Random bytes are: ")
Dim i As Integer
For i = 0 To 9
   Response.Write(i)
   Response.Write(":")
Response.WriteLine(b(i))
next
```

FUNCIONES FINANCIERAS

Además de las funciones matemáticas estándar, Visual Basic dispone de una serie de funciones por medio de las cuales podremos calcular el valor futuro de una inversión, la tasa interna de retorno, el interés o el capital pagado en un período.

Función	Descripción
DDB	Cálculo de la depreciación en un período de tiempo
FV	Valor futuro de unas aportaciones periódicas
IPmt	Interés pagado en un determinado periodo
IRR	Tasa interna de retorno
MIRR	Tasa interna de retorno modificada
NPer	Número de periodos de un pago constante
NPV	Valor actual de una inversión
Pmt	Importe de un pago periódico constante
PPmt	Capital pagado en un determinado periodo
PV	Valor actual de un pago futuro
Rate	Tipo de interés por periodo

2.- OPERACIONES CON CADENAS

Para realizar operaciones sobre cadenas de caracteres se usa la clase System.string con propiedades y métodos.

PROPIEDADES PÚBLICAS:

Propiedad	Descripción
Length	Obtiene el número de caracteres de la instancia en cuestión.

MÉTODOS PÚBLICOS:

Método	Descripción
Clone	Devuelve una referencia a la instancia de **String**.
Compare	Sobrecargado. Compara dos objetos **String** especificados.
CompareOrdinal	Sobrecargado. Compara dos objetos **String** sin considerar la referencia cultural ni el idioma nacional y local.
CompareTo	Sobrecargado. Compara la instancia en cuestión con un objeto especificado.
Concat	Sobrecargado. Concatena una o más instancias de **String** o las representaciones de tipo **String** de los valores de una o más instancias de Object.
Contains	Devuelve un valor booleano indicando si el objeto System.string especificado aparece dentro de esta cadena .
Copy	Crea una nueva instancia de **String** con el mismo valor que una **String** especificada.
CopyTo	Copia un número especificado de caracteres situados en una posición especificada de la instancia en una posición determinada de una matriz de caracteres Unicode.
EndsWith	Determina si el final de la instancia en cuestión coincide con el objeto **String** especificado.
Equals	Sobrecargado. Reemplazado. Determina si dos objetos **String** tienen el mismo valor.
Format	Sobrecargado. Reemplaza las especificaciones de formato de un objeto **String** especificado por el equivalente textual del valor de un objeto correspondiente.
GetEnumerator	Recupera un objeto que puede iterar los caracteres individuales de la instancia en cuestión.
GetHashCode	Reemplazado. Devuelve el código hash de esta instancia.
GetType (se hereda de Object)	Obtiene el objeto Type de la instancia actual.
GetTypeCode	Devuelve TypeCode para la clase **String**.
IndexOf	Sobrecargado. Devuelve el índice de la primera aparición de un objeto **String**, o de uno o más caracteres de la instancia en cuestión.

IndexOfAny	Sobrecargado. Devuelve el índice de la primera aparición en la instancia de un carácter de una matriz de caracteres Unicode especificada.
Insert	Inserta una instancia especificada de **String** en una posición de índice especificada de la instancia.
Intern	Recupera la referencia del sistema al objeto **String** especificado.
IsInterned	Recupera una referencia a un objeto **String** especificado.
Join	Sobrecargado. Concatena un objeto **String** separador especificado entre cada uno de los elementos de una matriz **String** especificada, generando una sola cadena concatenada.
LastIndexOf	Sobrecargado. Devuelve la posición de índice de la última aparición de un carácter Unicode especificado o de un objeto **String** en la instancia.
LastIndexOfAny	Sobrecargado. Devuelve la posición de índice de la última aparición en la instancia de uno o varios caracteres especificados de una matriz de caracteres Unicode.
PadLeft	Sobrecargado. Alinea a la derecha los caracteres de la instancia e inserta a la izquierda espacios en blanco o un carácter Unicode especificado hasta alcanzar la longitud total especificada.
PadRight	Sobrecargado. Alinea a la izquierda los caracteres de la cadena e inserta a la derecha espacios en blanco o un carácter Unicode especificado hasta alcanzar la longitud total especificada.
Remove	Elimina un número de caracteres especificado de la instancia a partir de una posición especificada.
Replace	Sobrecargado. Reemplaza todas las apariciones de un carácter Unicode o un objeto **String** en la instancia por otro carácter Unicode u otro objeto **String**.
Split	Sobrecargado. Identifica las subcadenas de la instancia que están delimitadas por uno o varios caracteres especificados en una matriz, y las coloca después en una matriz de elementos **String**.

StartsWith	Determina si el principio de la instancia coincide con el objeto **String** especificado.
Substring	Sobrecargado. Recupera una subcadena de la instancia.
ToCharArray	Sobrecargado. Copia los caracteres de la instancia en una matriz de caracteres Unicode.
ToLower	Sobrecargado. Devuelve una copia de **String** en minúsculas.
ToString	Sobrecargado. Reemplazado. Convierte el valor de la instancia en un objeto **String**.
ToUpper	Sobrecargado. Devuelve una copia de **String** en mayúsculas.
Trim	Sobrecargado. Quita todas las apariciones de un conjunto de caracteres especificados desde el principio y el final de la instancia.
TrimEnd	Quita todas las apariciones de un conjunto de caracteres especificados en una matriz de caracteres Unicode desde el final de la instancia.
TrimStart	Quita todas las apariciones de un conjunto de caracteres especificados en una matriz de caracteres Unicode desde el principio de la instancia.

3.- Operaciones Con Fechas

En C# contamos con una clase para poder realizar operaciones sobre fechas y horas que es la clase Datetime.

Sus propiedades y métodos son las siguientes:

Propiedades Públicas:

Propiedad	Descripción
Date	Obtiene el componente correspondiente a la fecha de esta instancia.
Day	Obtiene el día del mes representado por esta instancia.

DayOfWeek	Obtiene el día de la semana representado por esta instancia.
DayOfYear	Obtiene el día del año representado por esta instancia.
Hour	Obtiene el componente correspondiente a la hora de la fecha representada por esta instancia.
Millisecond	Obtiene el componente correspondiente a los milisegundos de la fecha representada por esta instancia.
Minute	Obtiene el componente correspondiente a los minutos de la fecha representada por esta instancia.
Month	Obtiene el componente correspondiente al mes de la fecha representada por esta instancia.
Now	Obtiene un **DateTime** que constituye la fecha y hora locales actuales de este equipo.
Second	Obtiene el componente correspondiente a los segundos de la fecha representada por esta instancia.
Ticks	Obtiene el número de pasos que representan la fecha y hora de esta instancia.
TimeOfDay	Obtiene la hora del día para esta instancia.
Today	Obtiene la fecha actual.
UtcNow	Obtiene un **DateTime** que representa la fecha y hora locales actuales de este equipo y que se expresa en forma de hora universal coordinada (UTC).
Year	Obtiene el componente correspondiente al año de la fecha representada por esta instancia.

MÉTODOS PÚBLICOS:

Método	Descripción
Add	Agrega el valor del TimeSpan especificado al valor de esta instancia.
AddDays	Agrega el número de días especificado al valor de esta instancia.
AddHours	Agrega el número de horas especificado al valor de esta instancia.
AddMilliseconds	Agrega el número de milisegundos especificado al valor de esta

	instancia.
AddMinutes	Agrega el número de minutos especificado al valor de esta instancia.
AddMonths	Agrega el número de meses especificado al valor de esta instancia.
AddSeconds	Agrega el número de segundos especificado al valor de esta instancia.
AddTicks	Agrega el número de pasos especificado al valor de esta instancia.
AddYears	Agrega el número de años especificado al valor de esta instancia.
Compare	Compara dos instancias de DateTime y devuelve una indicación de sus valores relativos.
CompareTo	Compara esta instancia con un objeto especificado y devuelve una indicación de los valores relativos.
DaysInMonth	Devuelve el número de días del mes especificado en el año especificado.
Equals	Sobrecargado. Reemplazado. Devuelve un valor que indica si una instancia de DateTime es igual a un objeto especificado.
FromFileTime	Devuelve un DateTime que equivale a la marca de hora del archivo del sistema operativo especificado.
FromOADate	Devuelve un DateTime que equivale a la fecha de automatización OLE especificada.
GetDateTimeFormats	Sobrecargado. Convierte el valor de esta instancia en todas las representaciones de cadena admitidas por los especificadores de formato DateTime estándar.
GetHashCode	Reemplazado. Devuelve el código hash de esta instancia.
GetType (se hereda de Object)	Obtiene el objeto Type de la instancia actual.
GetTypeCode	Devuelve TypeCode para el tipo de valor DateTime.
IsLeapYear	Devuelve una indicación en la que se precisa si el año especificado es bisiesto.
Parse	Sobrecargado. Convierte la representación de cadena

	especificada de una fecha y hora en su DateTime equivalente.
ParseExact	Sobrecargado. Convierte la representación de cadena especificada de una fecha y hora en su DateTime equivalente. El formato de la representación de cadena debe coincidir exactamente con un formato ya especificado.
Subtract	Sobrecargado. Resta la hora o duración especificada de esta instancia.
ToFileTime	Convierte el valor de esta instancia en el formato de la hora del archivo del sistema local.
ToLocalTime	Convierte la hora universal coordinada (UTC) actual en la hora local.
ToLongDateString	Convierte el valor de esta instancia en la representación de cadena de fecha larga equivalente.
ToLongTimeString	Convierte el valor de esta instancia en la representación de cadena de hora larga equivalente.
ToOADate	Convierte el valor de esta instancia en la fecha de automatización OLE equivalente.
ToShortDateString	Convierte el valor de esta instancia en la representación de cadena de fecha corta equivalente.
ToShortTimeString	Convierte el valor de esta instancia en la representación de cadena de hora corta equivalente.
ToString	Sobrecargado. Reemplazado. Convierte el valor de esta instancia en la representación de cadena equivalente.
ToUniversalTime	Convierte la hora local actual en la hora universal coordinada (UTC).