C#

-Empezar un programa

```
-class nombre del programa
{
}
```

Este es el inicio del programa,es completamente necesario poner class al principio del programa para que este funcione,las llaves marcan el inicio y el fin del programa.

```
-static void Main()
{
}
```

Este es el cuerpo del programa,la parte principal que contendrá los comandos del programa.

-Normas

-Cada orden en C# debe terminar en punto y coma(;).

-C# distingue entre mayúsculas y minúsculas,por lo que los comandos hay que escribirlos correctamente o sino no los detectará como tal.

-Declaración de variables

-**int** variable (valor entero)

-Se puede dar valor a la variable instantáneamente después de declararla.

Ejemplo

```
int primernumero = 5;
```

-Se pueden declarar varias variables del mismo tipo a la vez.

Ejemplo

```
int primernumero, segundonumero;
```

-O incluso ambas cosas a la vez

Ejemplo

```
int primernumero = 2, segundonumero = 5;
```

-**string** variable (texto).

-**float** variable (valor con decimales de baja precisión).

- **double** variable (valor con decimales de alta precisión).

-Comandos

```
System.Console.WriteLine()
```

Este comando sirve para escribir un mensaje en la consola,el texto que se quiera escribir se escribe entre comillas “ ”,también se pueden escribir valores u operaciones matemáticas,estas sin comillas,lo mismo pasa con las variables.

```
A =System.Convert.ToInt32(System.Console.ReadLine());
```

Convierte el valor introducido por el usuario(string/texto) a int,lo cual permite realizar operaciones con esos datos.

```
A=System.Convert.ToSingle(System.Console.ReadLine());
```

Convierte el valor introducido por el usuario(string/texto) a float,lo cual permite realizar operaciones más precisas con esos datos.

Ejemplo

mensaje con texto

```
System.Console.WriteLine("Hola Fran”);
```

Variables

```
System.Console.WriteLine(A);
```

mensaje con operaciones y valores numéricos

```
System.Console.WriteLine(2+8);
```

Combinación de ambas

```
System.Console.WriteLine("La suma de” + 2 + " y ” + 3 + "es”  + 5);
```

Otra forma de combinar ambas,especificando a posteriori donde va cada variable

```
System.Console.WriteLine("la suma de {0} y {1} es {2}”, primernumero, segundonumero, suma);
```

```
System.Console.Write()
```

Es como el comando WriteLine pero sin cambiar de línea.

```
-texto = System.Console.ReadLine()
```

Este comando sirve para hacer que el usuario sea quien teclee los valores.

Operadores

Operador	Operación
+	Suma
-	Resta,negación
*	Multiplicación
/	División
%	Resto de la división(“módulo”)

Orden de prioridad de los operadores

-En primer lugar se realizan las operaciones indicadas entre paréntesis .
-Luego la negación.
-Después las multiplicaciones,divisiones y el resto de la división.
-Finalmente,las sumas y las restas.
-En caso de tener igual prioridad,se analizan de izquierda a derecha.

Comentarios

hay dos formas de escribir comentarios en el código,los cuales el programa omitirá y no leerá,pero todo aquel que lea el código será capaz de ver,normalmente son detalles o indicaciones para entender mejor el código,estas son las 2 formas de realizarlos:

-Con 2 barras (//)

-Con /* *al principio y al final del comentario* */

Ejemplo

```
//este programa sirve para sumar
```

```
/*este programa sirve para sumar*/
```

Atajos

```
-using System
```

Este comando que se pone al principio del todo sirve para no tener que poner System cada vez,dado que ya estaría siendo usado desde el principio.

Estructuras de control

Estructuras alternativas

if

Esta construcción se emplea para realizarción. condicionales,en este caso el código solo se activaría si se cumple la condición.

```
if (condición)
{
}
```

Else

Esta orden podemos utilizarla para especificar que queremos que ocurra en caso de que no se cumpla la condición del if.

```
if (condición)
{
}
else
{
}
```

If-Else

Una forma de mantener el código de una forma más ordenada y sin tantas tabulaciones.

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

Operadores relacionales

Operador	Operación
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
==	Igual a
!=	Distinto a

Operadores lógicos

Operador	Significado
&&	Y
||	O
!	No

Tabla de la verdad para Y

A	B	A&&B
Falso	Falso	Falso
Falso	Verdadero	Falso
Verdadero	Falso	Falso
Verdadero	Verdadero	Verdadero

Tabla de la verdad para O

A	B	A||B
Falso	Falso	Falso
Falso	Verdadero	Verdadero
Verdadero	Falso	Verdadero
Verdadero	Verdadero	Verdadero

Tabla de la verdad del No

A	!A
Falso	Verdadero
Verdadero	Falso

Operador condicional ?

Es una forma abreviada de la estructura if else,si bien solo es útil para la asignación de valores.

```
nombreVariable = condicion ? valor1 : valor2;
```

Switch

se emplea cuando queremos analizar varios posibles valores(delimitados) de una variable o expresión.

```
switch(variable)
{
	case valor1:
		instrucciones;
		break;
	case valor2:
		instrucciones;
		break;
	…
	default:
		instrucciones;
		break;
```

si dos o más casos hacen lo mismo,podemos ponerlos consecutivos y dejarlos vacíos.

```
switch(variable)
{
	case valor1:
	case valor2:
	case valor3:
	case valor4:
		instrucciones;
		break;
	case valor5:
		instrucciones;
		break;
	…
	default:
		instrucciones;
		break;
```

While

Se emplea para crear bucles,permite realizar un conjunto de instrucciones mientras se cumpla la condición marcada.

```
while (condicion)
{
	instruccion1;

	…
	instruccionN;
}
```

Do While

La diferencia con while es que con esto la instrucción se realiza por lo menos una vez antes de evaluar la condición del while.

```
do
{
	instruccion1;

	…
	instruccionN;
}
while (condicion);
```

For

Permite ejecutar un conjunto de instrucciones mientras se cumpla una determinada condición, o un número determinado de veces,sirve especialmente para hacer contadores.

```
for (variable; condición; incremento)
{
      instruccion1;

      …
      instruccionN;
}
```

Autoincremento y Autodecremento

Los operadores ++ y –- sirven para autoincrementar/autodisminuir la variable sobre la que se aplican.

Bucles infinitos

Bucles con una condición que siempre se cumple,se usa en aplicaciones que requieren de un evento externo para terminar.

```
// Bucle while infinito
while (1 == 1)
{
      …
}

// Bucle for infinito
for (;;)
{

      …
}
```

Bucles anidados

Básicamente son bucles dentro de bucles,empleado para resolver tareas más complejas.

Break

Se utiliza para salir inmediatamente del bucle en el que nos encontramos.

Continue

Se utiliza para omitir las siguientes instrucciones del bucle.

GoTo

Se emplea para saltar directamente a una etiqueta en el código.

```
for (int i = 1; i <= 10; i++)
{
        if (i == 5)
                goto siguiente;
 }
siguiente:
 Console.WriteLine("Fin");
```

Foreach

Permite recorrer automáticamente todos los elementos de una colección.

```
Console.Write("Dime tu nombre: ");
string nombre = Console.ReadLine();
foreach(char letra in nombre)
{
        Console.WriteLine(letra);
}
```

uso de excepciones

TRY-CATCH

```
int dividendo, divisor;

try
{
        Console.WriteLine("Dime el dividendo:");
        dividendo = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("Dime el divisor:");
        divisor = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("La división es: {0}",
        (dividendo / divisor));
}
catch (Exception e)
{
        Console.WriteLine("Error: no se puede” + “dividir
        por cero")
        Console.WriteLine(e.Message);
```

Tipos de dato entero

Tipo	Tamaño(bytes)	Rango
sbyte	1	-128 a 127
byte	1	0 a 255
short	2	-32768 a 32767
ushort	2	0 a 65535
int	4	-2147483648 a 2147483647
uint	4	0 a 4294967295
long	8	núms de 19 cifras
ulong	8	núms de 19-20 cifras

Conversiones de cadena a entero

Convert.ToByte	Convert.ToSByte
Convert.ToSByte	Convert.ToInt16
Convert.ToUInt32	Convert.ToInt32
Convert.ToUInt64	Convert.ToInt64

Incremento y decremento

-El operador ++ incrementa 1 unidad el valoe de la variable en la que se aplicaciones.

-El operador -- decrementa 1 unidad el valoe de la variable en la que se aplicaciones.

Preincremento y postincremento

Incremento

Post-incremento
-si aplicamos el operador ++ detrás de una variable en una expresión, primero se calcula la expresión y luego se cambia el valor de la variable

Pre-incremento

- si lo aplicamos delante, primero se cambia el valor de la variable y luego se evalúa la expresión

Decremento

Post-decremento
-si aplicamos el operador -- detrás de una variable en una expresión, primero se calcula la expresión y luego se cambia el valor de la variable

Pre-decremento

- si lo aplicamos delante, primero se cambia el valor de la variable y luego se evalúa la expresión

Operaciones abreviadas

Operación	Significado
+=	Autosuma
-=	Autoresta
*=	Automultiplicación
/=	Autodivisión
%=	Autoresto

Desbordamiento

Algunas operaciones con un tipo de datos pueden provocar que se exceda del rango permitido.
Se produce lo que se llama **desbordamiento** (*overflow*)

Cambios de base

```
Convert.ToString (número,base)
```

Binario

```
Convert.ToString (número,2)
```

Hexadecimal

```
Convert.ToString (número,16)
```

TIPOS REALES

Tipo	Tamaño(bytes)	Significado
float	4	Simple precisión (7-8 cifras significativas)
double	8	Doble precisión (15-16 cifras sign.)
decimal	16	28-29 cifras sign.

Declaración de variables reales

-El separador decimal es el punto(.)

-Al definir un valor de tipo **float** se debe finalizar con una f para que el compilador lo reconozca como **float**

Pedir datos reales al usuario

-Convert.ToSingle para tipo float

-Convert.ToDouble para tipo double

-Convert.ToDecimal para tipo decimal

Typecast

Fuerza la conversión de un tipo de dato a uno similar(entero a real y viceversa)

Se pone entre paréntesis el tipo de dato al que se quiere convertir la variable o expresión

ejemplo

```
float resultado = (float)n1 / n2;
```

Formatear números

-Podemos decidir cuantos dígitos se muestran en la parte entera y en la parte decimal.

-Empleamos la instrucción **.ToString** del dato que queremos mostrar

Símbolos de formato

Símbolo	Significado
0	Debe aparecer un dígito, o 0 en caso contrario
#	Mostrar dígito si lo hay, o nada si no
.	Separador parte entera y decimal
N1	Mostrar 1 decimal
N2	Mostrar 2 decimales
N3	Mostrar 3 decimales

Funciones matemáticas

Math.Abs(x)	valor absoluto de x
Math.Sqrt(x)	raíz cuadrada de x
Math.Pow(x, y)	x elevado a y (potencia)
Math.Ceiling(x)	redondeo por arriba de x
Math.Floor(x)	redondeo por debajo de x
Math.PI	Constante matemática PI
Math.E	Constante matemática E

Otros tipos básicos

Carácteres

-Se representan con el tipo **char**

-Los valores se representan con comillas simples(‘ ’)

-Podemos usar **Convert.ToChar** para leer de teclado

-Los caracteres se codifican con un número asignado

Secuencias de escape

(Caracteres especiales que representan ciertos símbolos)

Símbolo	**Significado**
\n	Salto de línea
\t	Tabulación
\"	Comilla doble
\'	Comilla simple
\\	Barra invertida

Texto

-Tipo **string**

-Los valores se representan con comillas dobles(“ ”)

-podemos usar **Console.ReadLine** directamente para leer de teclado

Booleano

-Se representa con el tipo **bool**

-Puede tomar tomar los valores de true o false

Constantes

-Datos cuyo valor no cambia a lo largo de la ejecución del programa

-Se define igual que con las variables,pero poniendo

```
const int A;
```

Enumeraciones

-Representa un conjunto finito de valores

-Se usa **enum** seguida del nombre de la enumeración y de los valores entre llaves.

-Se declara fuera del main

```
enum diasSemana {
 LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO
};
```

Variables de tipo implícito

-Usamos el tipo **var** cuando no sepamos de antemano de que tipo va a ser la variable

-PROHIBIDO usarlo salvo que se indique explícitamente en el ejercicio