**FUNCIONES**

Hasta ahora hemos estado pensando los pasos que deberíamos dar para resolver un cierto problema, y hemos creado programas a partir de cada uno de esos pasos. Esto es razonable cuando los problemas son sencillos, pero puede no ser la mejor forma de actuar cuando se trata de algo más complicado.

A partir de ahora vamos a empezar a intentar descomponer los problemas en trozos más pequeños, que sean más fáciles de resolver. Esto nos puede suponer varias ventajas:

* Cada "trozo de programa" independiente será más fácil de programar, al realizar una función breve y concreta.
* El "programa principal" será más fácil de leer, porque no necesitará contener todos los detalles de cómo se hace cada cosa.
* Evitaremos mucho código repetitivo.

Esos "trozos" de programa son lo que suele llamar "subrutinas", "procedimientos" o " **funciones** ".

Definicion

 forma básica de **definir** una función será indicando su **nombre** seguido de unos **paréntesis** vacios (o no) y precediéndolo por palabras reservadas, como "public static **void**",

Después, entre llaves indicaremos todos los pasos que queremos que dé ese "trozo de programa".

public static void Saludar()

{

Console.Write("Bienvenido al programa ");

Console.WriteLine("de ejemplo");

}

Ahora desde dentro del cuerpo de nuestro programa, podríamos "**llamar**" a esa función:

public static void Main()

{

Saludar();

...

}

Un detalle importante: tanto la función habitual "Main" como la nueva función "Saludar" serían parte de nuestra "class", es decir, son independientes, (no están ninguna dentro de la otra)

Ejemplo:

**LeerDatosDeFichero**();

do {

**MostrarMenu**();

opcion = **PedirOpcion**();

switch( opcion ) {

case 1: **BuscarDatos**(); break;

case 2: **ModificarDatos**(); break;

case 3: **AnadirDatos**(); break;

...

Ejercicio:

Crea una función llamada "DibujarCuadrado10x10" y otra función llamada “borrar3lineas”, la primera dibuja un cuadrado formato por 10 filas con 10 asteriscos cada una, la segunda creara 3 lineas en blanco. En el "Main" llamara tres veces a cada función

**Funciones con parametros**

Es muy frecuente que nos interese indicarle a nuestra función ciertos datos con los que queremos que trabaje.

Los llamaremos "parámetros" y los indicaremos dentro del paréntesis que sigue al nombre de la función, separados por comas. Para cada uno de ellos, deberemos indicar su tipo de datos (por ejemplo "int") y luego su nombre.

public static void EscribirSuma( int a, int b )

{ ...

}

El valor "**void**" Cuando queremos dejar claro que una función no tiene que devolver ningún valor.

public static int Suma ( int a, int b )

{

**return** a+b;

}

Ejercicio

Programa que me pida un número y me devuelva el factorial del mismo

***Variables locales y variables globales***

Hasta ahora, hemos declarado las variables dentro de "Main". Ahora nuestros programas tienen varios "bloques", así que las variables se comportarán de forma distinta según donde las declaremos.

Las variables se pueden declarar dentro de un bloque (una función), y entonces sólo ese bloque las conocerá, no se podrán usar desde ningún otro bloque del programa. Es lo que llamaremos "**variables locales**".

Por el contrario, si declaramos una variable al comienzo del programa, fuera de todos los "bloques" de programa, será una "**variable global**", a la que se podrá acceder desde cualquier parte.

(Por ahora, una variable global deberá llevar siempre la palabra "static")

En general, deberemos intentar que la mayor cantidad de variables posible sean locales (lo ideal sería que todas lo fueran).

La forma correcta de pasar datos entre distintos trozos de programa no es a través de variables globales, sino usando los parámetros de cada función y los valores devueltos

***Modificando parámetros. Parámetro por referencia***

Podemos modificar el valor de un dato que recibamos como parámetro, pero posiblemente el resultado no será el que esperamos. Vamos a verlo con un ejemplo:

Supongamos el siguiente código

public static void Duplicar(int x)

{

Console.WriteLine(" El valor recibido vale {0}", x); -> 5

x = x \* 2;

Console.WriteLine(" y ahora vale {0}", x); -> 10

}

public static void Main()

{

int n = 5;

Console.WriteLine("n vale {0}", n); -> 5

Duplicar(n);

Console.WriteLine("Ahora n vale {0}", n); -> 5

}

Vemos que al salir de la función, no se conservan los cambios que hagamos a esa variable que se ha recibido como parámetro.

Si quisiéramos que la variable pasada como parámetro **SI** conserve los cambio tengo que hacerlo pasando dichos parámetros "**por referencia**", lo que se indica usando la palabra "**ref**", tanto en la definición de la función como en su llamada

public static void Duplicar(**ref** int x)

{

Console.WriteLine(" El valor recibido vale {0}", x); -> 5

x = x \* 2;

Console.WriteLine(" y ahora vale {0}", x); -> 10

}

public static void Main()

{

int n = 5;

Console.WriteLine("n vale {0}", n); -> 5

Duplicar(**ref** n);

Console.WriteLine("Ahora n vale {0}", n); -> 10

}

Esto lo podríamos hacer con mas de un parámetro.

public static void Intercambia(**ref** int x, **ref** int y)

Además de pasar parámetros por valor y por referencia en C# existe una posibilidad adicional que no existe en otros lenguajes, como los "**parámetros de salida**". Esto se hace con la palabra **out**

La diferencia es: los parámetros pasados como ref deben estar inicializados previamente (se va a modificar su valor), mientras que los parámetros pasados como out no es necesario que lo estén (se va a asignar un valor sin importarnos lo que tuviera previamente).

public static void assignaValor(out int parametro1, out int parametro2)

{

parametro1 = 14\*2;

parametro2 = 23\*2;

}

public static void Main()

{

int num1, num2;

assignaValor(out num1, out num2);

Console.WriteLine("Numero 1 {0} Numero 2 {1} ", num1, num2);

Console.ReadLine();

}

**Ejercicios propuestos:**

Crea una función "Intercambiar", que intercambie el valor de los dos números enteros que se le indiquen como parámetro. Crea también un programa que la pruebe.

Dime numero 1: 45

Dime numero 2: 56

El numero 1 es 45 y el 2 es 56

Llamais a la función

El numero 1 es 56 y el 2 es 45

**Algunas funciones útiles de cadenas**

**Substring**, **extraer** parte del contenido de una cadena

saludo = frase.Substring(0,4);

Recibe dos parámetros: la posición a partir de la que queremos empezar y la cantidad de caracteres que queremos obtener. Si omitir el segundo número, y entonces se extraerá desde la posición indicada hasta el final de la cadena.

**IndexOf busca**  una cadena

nombre.**IndexOf**("Juan")

Para ver si una cadena contiene un cierto texto, podemos usar IndexOf ("posición de"), que nos dice en qué posición se encuentra, siendo 0 la primera posición (o devuelve el valor -1, si no aparece)

Podemos añadir un segundo parámetro opcional, que es la posición a partir de la que queremos buscar:

if (nombre.IndexOf("Juan", 5) >= 0) ...

De forma similar, **LastIndexOf** ("última posición de") indica la última aparición (es decir, busca de derecha a izquierda).

Si solamente queremos ver si aparece, pero no nos importa en qué posición está, nos bastará con usar "**Contains**":

if (nombre.Contains("Juan")) ...

Algunas más

**ToUpper()** convierte a mayúsculas:

nombreCorrecto = nombre.ToUpper();

**ToLower()** convierte a minúsculas:

password2 = password.ToLower();

**Insert(int posición, string subcadena):** Insertar una subcadena en una cierta posición de la cadena inicial:

nombreFormal = nombre.Insert(0,"Don");

**Remove(int posición, int cantidad):** Elimina una cantidad de caracteres en cierta posición:

apellidos = nombreCompleto.Remove(0,6);

**Replace(string textoASustituir, string cadenaSustituta)**: Sustituye una cadena (todas las veces que aparezca) por otra:

nombreCorregido = nombre.Replace("Pepe", "Jose");

**CompareTo**: Compara cadenas, al giaul que el > en numero

if (frase.CompareTo("hola") > 0) Console.WriteLine("La frase es mayor que hola");

Si NO quisiera distinguir entre mayúscula y minúscula

if (String.Compare(frase, "hola", true) > 0) Console.WriteLine("Es mayor que hola (mays o mins)");

**Algunas funciones útiles de numeros**

**Numeros aleatorios**: debemos crear un objeto de tipo "**Random**" (una única vez), y luego llamaremos a "**Next**" cada vez que queramos obtener valores entre dos extremos:

// Creamos un objeto Random

**Random** generador = **new Random();**

// Generamos un número entre dos valores dados // (el segundo límite no está incluido)

int aleatorio = generador.**Next**(1, 101);

Funciones matematicas. Todas ellas se usan precedidas por "**Math**."

**Abs**(x): Valor absoluto

**Acos**(x): Arco coseno

**Asin**(x): Arco seno

**Atan**(x): Arco tangente

**Atan2**(y,x): Arco tangente de y/x (por si x o y son 0)

**Ceiling**(x): El valor entero superior a x y más cercano a él

**Cos**(x): Coseno

**Cosh**(x): Coseno hiperbólico

**Exp**(x): Exponencial de x (e elevado a x)

**Floor**(x): El mayor valor entero que es menor que x

**Log**(x): Logaritmo natural (o neperiano, en base "e")

**Log10**(x): Logaritmo en base 10

**Pow**(x,y): x elevado a y

**Round**(x, cifras): Redondea un número

**Sin**(x): Seno

**Sinh**(x): Seno hiperbólico

**Sqrt**(x): Raíz cuadrada

**Tan**(x): Tangente

**Tanh**(x): Tangente hiperbólica

* La raíz cuadrada de 4 se calcularía haciendo x = Math.Sqrt(4);
* La potencia: para elevar 2 al cubo haríamos y = Math.Pow(2, 3);
* El valor absoluto: para trabajar sólo con números positivos usaríamos n = Math.Abs(x);