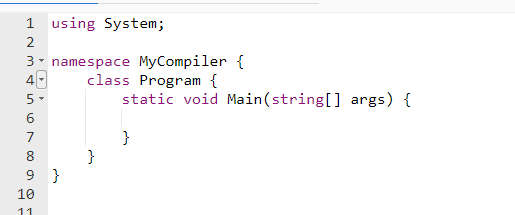
# Datos importantes

Todo código en C# debe llevar esta estructura básica.



# Comentarios

// (doble slach) sierve para realizar comentarios en una sola línea

/\* (slach multiplicación signo) sirve para comentar una o más líneas

Comentar sirve para que al leer el algoritmo un tercero este sepa que se realiza en tal línea o bloque de código

# Identificadores

Son los nombres que se usan para identificar los elementos del programa:

* namespaces
* clases
* métodos
* variables
* constantes

# Reglas o convenciones

* Solo se pueden usar:
  + letras (mayúsculas o minúsculas)
  + números
  + guiones bajos (“\_”)

ej.

Primera\_Aplicacion (correcto)

Primera-Aplicación (incorrecto)

* las definiciones de variables, clases, namespaces, contastantes deben comenzar por una letra o un guion bajo

ej.

PrimeraAplicacion2 (correcto)

2primeraApliacion (incorrecto)

* no se deben utilizar palabras claves o reservadas para nombrar variables, namespaces, clases, contantes

***def palabras claves o reservadas: son las que ocupa el lenguaje para sus propios propósitos***

ej.

Class namespaces (incorrecto)

Class algo (correcto)

# Buenas prácticas

1. lo ideal es dejar un espacio vertical entre bloques, es decir un espacio entre cada línea de código.
2. Tabular el código por bloques

## Variables

1. No comenzar el nombre de una variable con un guion bajo
2. No crear más de una variable que se diferencien por una sola letra.

Ej. Edad, edad, EDAD

1. Comenzar el nombre de una variable por minúsculas
2. Si el nombre de la variable está compuesto por más de una palabra es recomendado usar la estructura “CAMEL CASE”. EJ radioDelCirculo
3. No utilizar notación húngara, esto se refiere a no definir las variables de la siguiente manera “bedad” que es la mezcla del tipo de dato byte y el nombre de una variable en este caso la variable edad

# Tipos de datos

Existen dos grandes tipos de datos estos son los por valor y los por referencia

Por valor tenemos a los siguientes tipos de datos

Por valor:

* Enteros:

Estos a su vez se dividen en:

* + Sin signo:
    - Byte
    - Ushort
    - Uint
    - Ulong
  + Con signo:
    - Sbyte
    - Short
    - Int
    - Long
* Reales

Son todos los números que incluyan una parte decimal

* + float
  + double
  + decimal
* booleanos:

Valores lógicos para toma de decisiones dentro del programa

* + True : para valores verdaderos
  + False: para valores falsos
* Letras

Son como dice su nombre todos los caracteres de tipo str o char (UTF 16)

# Variables

La declaración de una variable en C# se realiza de la siguiente manera y siempre se termina la ninea con un punto y coma “;”

Tipo de dato + el nombre de la variable;

Ejemplo

Int edad;

Al declarar una una variable esto no quiere decir que esta inicializada para realizar esto bien se puede realizar en la misma line que se define o en algún otro lugar del programa.

Ej

Declaración e inicialización

Int edad = 28;

Declaración y posterior inicialización

Int edad;

Edad = 28;

***Consideración: en C# no se pueden utilizar variables no inicializadas.***

# Operadores aritméticos

Operaciones básicas

* Suma (+) puede ser utilizado tanto para números como para str
* Resta (-) puede ser utilizado tanto para números como para str
* Multiplicación (\*)
* División (/)
* Modulo (%)

Incremento (++)

Decremento (--)

Conjugaciones de símbolos:

* Suma (+) 🡪 +=
* Resta (-) 🡪 -=

Nota:

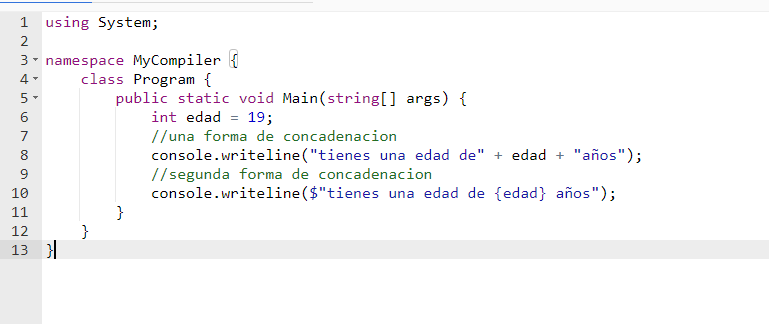
Al dividir el en C# siempre entregara el resultado en el tipo de dato esta la variable o el número.

Ej al dividir 5 en 2 debería dar 2.5 pero en C# nuestra por consola 2, esto debido a que ambos son números enteros por tanto el lenguaje respeta esto, a diferencia de otros lenguajes

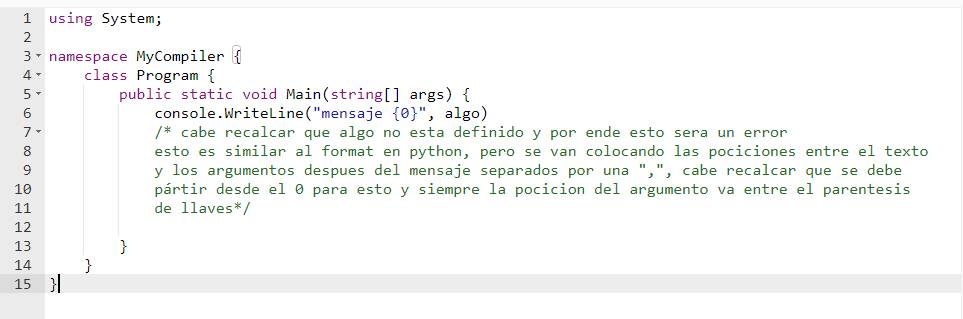
Entero/entero = entero

Entero/decimal = decimal

Concatenación de string

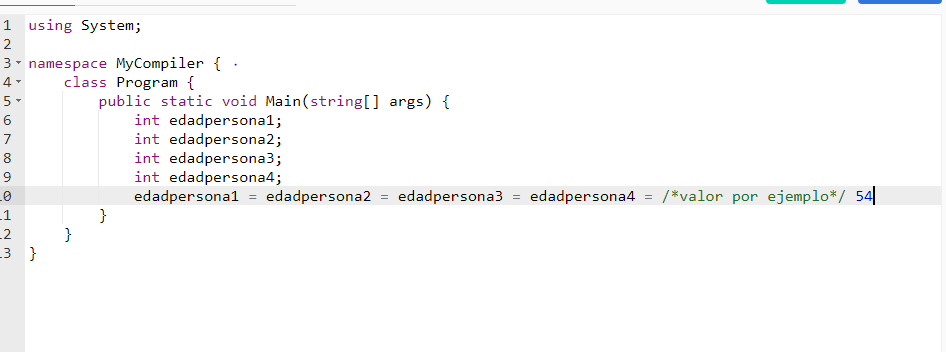


Otra forma de realizar la impresión de datos por pantalla al usuario es:



Comportamiento “=”

Puede servir para asignar un valor a múltiples variables



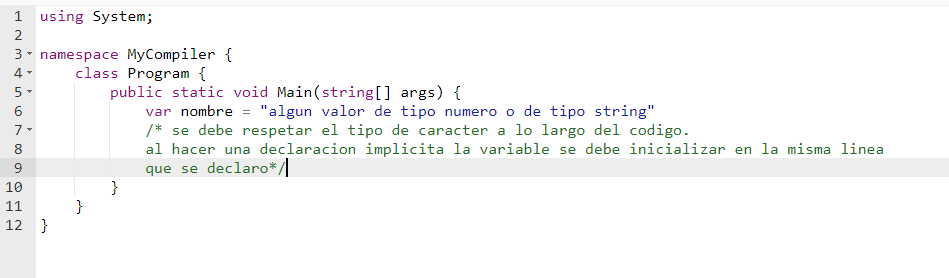
No se recomienda hacer esto de esta forma ya que conduce a errores

Int edadpersona1, edadpersona2, edadpersona3, edadpersona4 = 27;

Ya que la única variable que se inicializa es edadpersona4 y las otras 3 no

# Declaración implícita

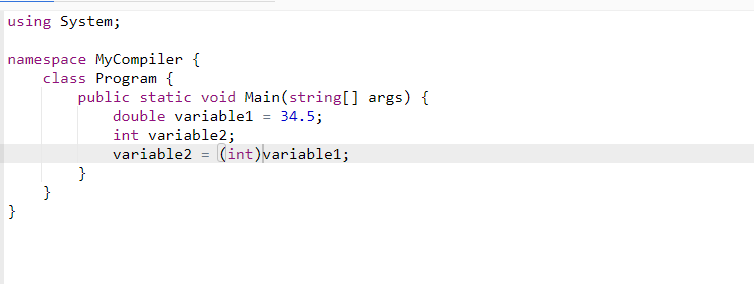
*Se suele confundir con visual basic el lenguaje de trabajo en Excel*

**

# Conversiones entre tipos de datos

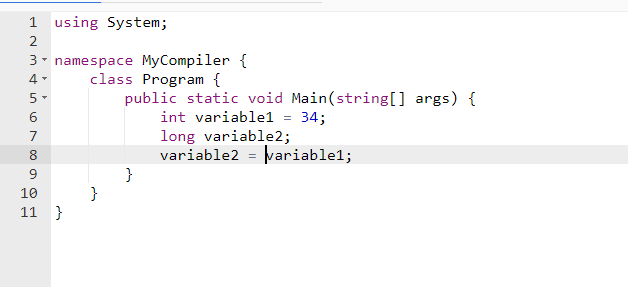
## Conversión explicita/ casting

Es la conversión que se realiza entre dos tipos de datos incompatible por ejemplo de double a int



## Conversión implícita

Conversión de tipos de datos compatibles entre si, pero de diferente alcance



Agregar tabla de compatibilidad

Implícita

Explicita

# Conversión de tipos no compatibles entre si

Conversión de str a num

Int.parse() permite pasar de un str a números, ya que al solicitar un valor por pantalla al usuario este será de tipo str y para operar con este es necesario pasarlo a numero

El método parce se puede usar con cualquier tipo de número con la siguiente estructura

TipoDeNumero.Parse ()

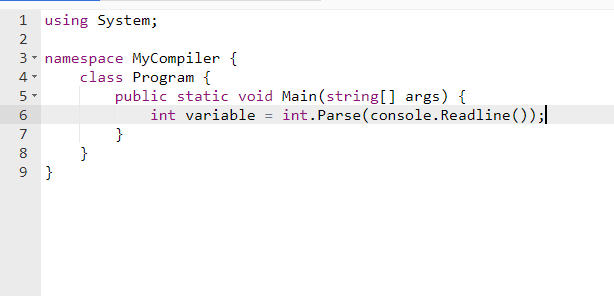
Solicitar datos por pantalla

Se usa el método console.ReadLine () para recibir datos del usuario pero estos independiente del tipo que este sea se guarda como un str en la variable siempre

Usando ambos métodos el parse y RedLine se pueden usar varios tipos de datos

Por ejemplo

Se pueden recibir números para operar con ellos



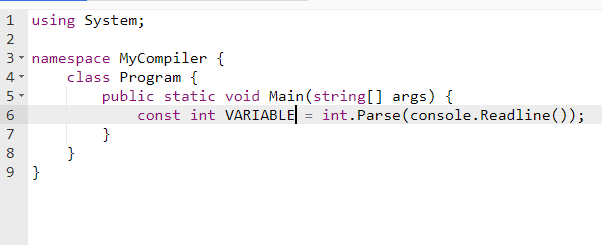
# Constante

Espacio en la memoria del ordenador sonde se almacena un valor que no podrá cambiar durante la ejecución del programa

Definición:

Const “tipoDeDato” nombre = Valor(str/numérico);

Las constantes se deben declarar e inicializar en la misma línea. También se suele colocar el nombre de la constante por convención.



# Métodos

## Que son los métodos

Son un grupo de sentencias (instrucciones) a las que se les da un nombre identificado, que realiza una tarea en concreto

## Para que sirve

Para realizar una tarea concreta en un momento determinado. Un método no realiza nada hasta que es llamado en el Main. Este puede ser usado tantas veces se considere necesario su uso.

## Sintaxis

tipoDeDato nombreDelMetodo (parámetros) {

Cuerpo del método

}

TipoDeDato: este poder ser int, double, string, o cualquier tipo de dato valido en C#

nombreDelMetodo: sigue las mismas reglas que las variables y constantes *(ver apartado de buenas prácticas)*

Cuerpo del método: son todas instrucciones que va a realizar un método por ejemplo sumar n números

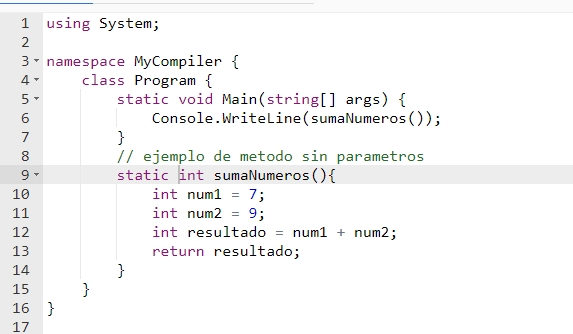
Al definir un método sin la palabra “static” solo dará un error de objeto, pero no momento solo colocaremos “static” aunque no entendamos muy bien lo que es. Esto lo veremos en la parte de programación orientada a objetos.

Consideraciones:

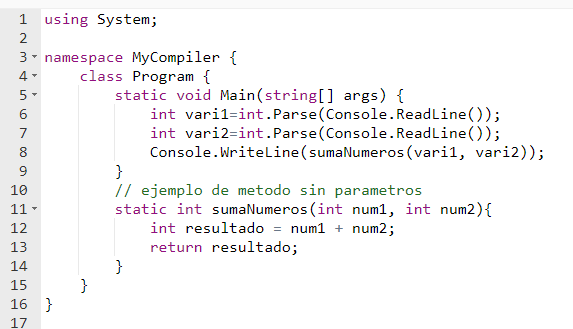
* No se ejecuta hasta que es llamado
* Todos los métodos deben ir dentro de una clase
* Se debe especificar el tipo devuelto y los parámetros
* En C# no hay distinción entre métodos y funciones ambos términos son lo mismo
* Siempre todos los tipos de datos deben llevar “return” a excepción del “void” porque de lo contrario dará error de compilación

Ejemplos

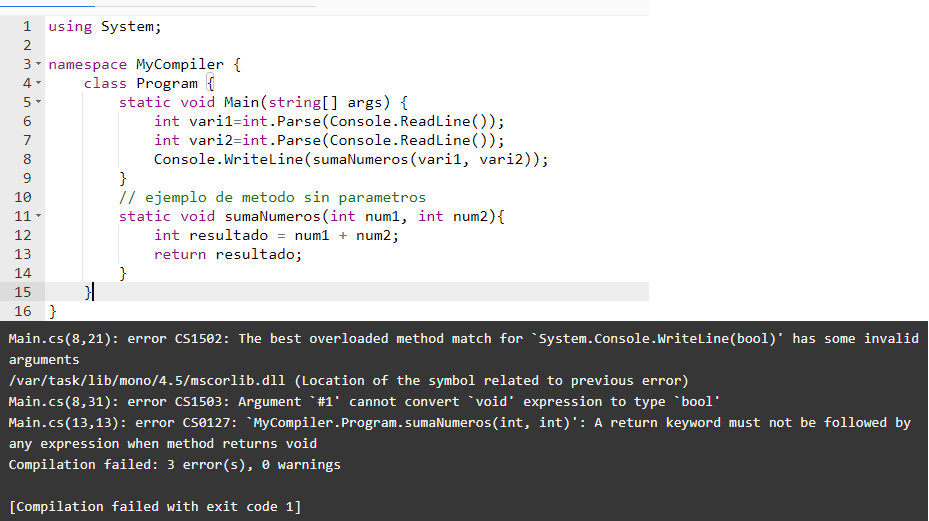
1. Por valor: con valor definido por el programador



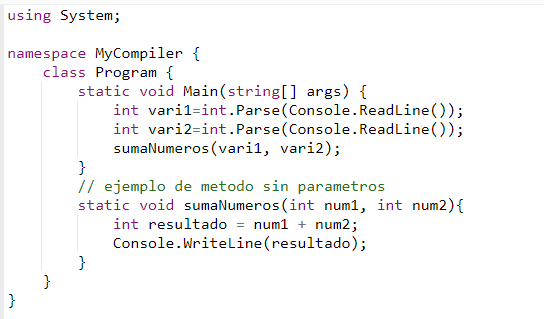
1. Por referencia: valor dado por el usuario del algoritmo



Para facilitar un poco el entendimiento de los métodos es recomendable pensar que son como una máquina expendedora de dulces o bebidas, ya que al ingresarle un número en el teclado esta te devuelve el producto asignado a ese numero

Al realzar un método con el “tipo de dato void” este no puede llevar un return ya que esta función no puede devolver ningún valor y al colocar un return dará error de compilación 

Uso correcto del método void



La llamada de un método se debe realizar dentro del método Main para que este funcione o de lo contrario no se realizara ninguna acción como se muestra en las imagines anteriores.

Da igual como estén los métodos ya sea antes o después del Main ya que se irán llamando con respecto al orden en que estén en el Main.

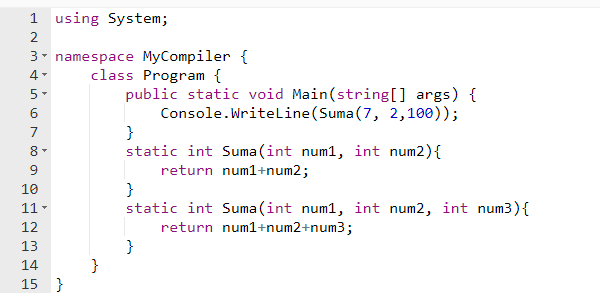
# Ámbito o alcance de métodos y variables

Al definir una variable dentro de un método esta solo es utilizable dentro del método, pero surgen ocasiones donde es necesario usar estas variables en más de un método y la solución a este problema es definir las variables en la clase. Al trabajar con variables definidas en la clase están son llamadas “campos de clase”.

***Sacar pantallazo***

## Sobrecarga de métodos

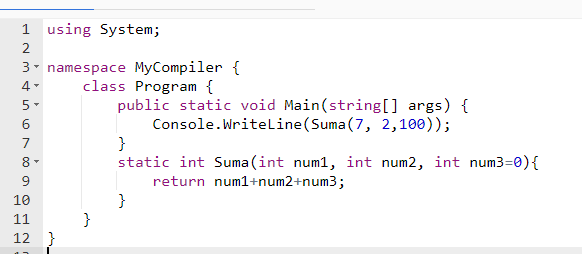
Es tener más de un método con el mismo nombre pero con distinto número de parámetros o que devuelvan un tipo de dato distinto.



El programa dependiendo de la cantidad de argumentos dados al método elegirá el mejor.

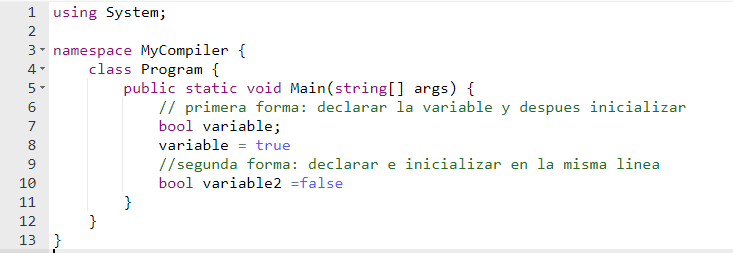
## Método con valor opcional

Es de ayuda cuando hora que c# trabaje con otros lenguajes de programación que no admitan la sobrecarga de metodos, es tan simple como colar un valor cualquiera a una parámetro del método pero siempre los parámetros opcionales deben estar al final.



# Condicionales

## Declaración de variables booleanas



Basta decir que estas variables solo pueden tomar como valores *True* o *False*

## Operadores booleanos

### Operadores de comparación

|  |  |  |
| --- | --- | --- |
| Símbolo | Nombre | Significado |
| ***==*** | Igual que | Compra que dos valores sean iguales |
| **!=** | Distinto de | Compra que dos valores sean distintos |
| **<**  **<=** | Menor que  Menor igual | Compara que el valor se menor estricto con el que se está comparando  Permite que el valor llegue a ser igual al con que comparamos |
| **>**  **>=** | Mayor que  Mayor igual | Compara que el valor se mayor estricto con el que se está comparando  Permite que el valor llegue a ser igual al con que comparamos |

### Operadores lógicos

|  |  |  |
| --- | --- | --- |
| Símbolo | Nombre | Significado |
| **!** | negación | Invierte el valor de una variable booleana |
| && | Y lógico | Ambas condiciones tienen que ser verdad para avanzar |
| || | O lógico | Por lo menos una de las condiciones debe ser verdadera |

Estos operadores son útiles cunado el programa debe tomar decisiones “por sí mismo”

## Sintaxis

Estructuras

If (condición){

Código a ejecutar en caso que la condición sea true

}

else if (condición){

Código a ejecutar en caso que la primera condición sea false y esta sea verdadera

}

else{

Código a ejecutar en caso que ningún de las condiciones anteriores sea verdadera

}

