Tipos de valor y referencia
clases y estructuras son tipos
Números son de valor, referencias son punteros a pila.
string es de referencia.

Valor → char
→ bool
→ números → enteros
→ flotantes

Referencias → Clases → object (referencia) → raíz de jerarquía de tipos
→ Delegaciones
→ interfaces.

Las q' están por ref. no se crean en la heap hasta que no le metes algo.

- conversiones de tipos
- Implícitas (int := bool) → Cuando no hay riesgo
- explícitas
- Métodos auxiliares → (int.Parse(string))
  → System.Convert
  → ToString → todos
- Definidas por el user los tipos lo considera

```
byte b = 10;
double x = 12.25;
int i = b;
double y = i;
short j = i;
i = x;
```

int i = b; → conversión implícita byte a int
double y = i; → conversión implícita int a double
short j = i; i = x; } error: No se puede hacer conv. implícita (int a short) double a int

Ojo: da problema de ejecu. { short J = (short) I
I = (int) x

Si el nro es + grande de lo q' entra.

string i = int(true)

Ojo con pérdida de precisión:
Puedo hacer de int a float implícitamente pero en nros muy grandes pierdo prec.
ej: int I := 100 000 000 038
float j = I → q' era con 100 000 000 000.

Object
→ console
→ string
→ value type → int, char, double, Enum

Aunque fuertemente tipado x jerarquía de tipos variables de tipo object admiten valores de cualquier tipo.
ej: int i = 123;
object = i; } Lo de arriba puede tener cosas de abajo, no al revés

por sistema unificado de tipos
boxing - unboxing → asignar variables de tipo valor a tipo referencia y al revés

object o;
int i = 123;
o = i; → cuando asigno tipo valor a ref se hace boxing
↓
almacena en heap en lugar de pila
(pila se queda con referencia del objeto q' está ...)
int j = (int)o

o = 1;
int j = (int)o → ref. a valor hay unboxing → valor q' estaba en heap queda en pila.

almacena en heap en lugar de pila (pila se queda con referencia del objeto q' está en heap)

NULL

para q' una var pueda admitir Null, tengo q' declararla con un ?

Ej: int? j = Null
string? pepe = Null

```
string? str1 = null;
string? str2 = "Hola";
```

pila: str1 0x000000; str2 0x12345 → heap: "Hola"

Has value - value.

```
int? i1 = 10;
int? i2 = null;
```

pila:
s1 { True → has value; 10 → value }
s2 { False → has value; 0 → value }

- i = j ?? -1; si no es null asigna el valor, sino asigna -1
- s = j.HasValue ? j.value : -1 → s = -1 (sino); s = j.value
- s = j != Null ? (int)j : -1 — j <> null? si true s = j, sino -1

Operador 'is'

- Consulto x tipo de una var.

```
Object o = 1
Console.WriteLine(o.is int) → True.
```

Operador "as"

- Se usa en casos que aceptan null conversión explícita.

Cuando no se puede hacer la conversión, en lugar de dar error da Null

E as T donde E devuelve un valor y T es nombre de un tipo

E is T ? (E)(T) : Null

```
Object obj = "casa";
string? st = (string) obj;
obj = 12;
```

→ tengo q' castearlo pq podría meter algo q' no es string en string.

```
st = obj as string → va a dar null (pq no puedo pasar int a string)
st = (string) obj; → tira error de ejecución
```

Variables locales

- C# no te permite leer una var que tiene basura.

no compila

Arreglos

int [] vector1;
vector1 = new vector1;

int [] vector1 = new int [3] {5, 1, 3}
" " = new int [] {5, 1, 3}
" " = {5, 1, 4}

Puede usar var para asignar tamaño al arreglo

int [in.length -1] → para acceder a ult. dato.
.length de tamaño, no cuánto está lleno.

FOREACH

- estructura de control. Recorre toda la estructura.
- foreach (int num in vector)
  ↳ var que va a contener lo de! luego.

Strings

- el x referencia pero no se comparan punteros al hacer == sino q se compara lo de adentro
- No se puede modificar strings

→ ST = ST + "mundo" → Crea una nueva string, no usa la vieja.
Va haciendo copias.

String builder

- como una string pero con + opciones

mutable

string builder STB = new StringBuilder ("C sharp")
Console.WriteLine(STB).

Tipos enumerativos

enum Tamaño {Chico, mediano, grande}
Tamaño T;
T = Tamaño.grande
T = (Tamaño) 0; // valor tamaño. chico ahora.

Metodos

Bloque con nombre de código ejecutable
se invoca
int sumar (int a, int b)
int res = a+b
return res;

Pasaje de valor → se asignan en el cuerpo del método
se pueden pasar por. de valor no inicializados.

void sumar (int a, b; out int res)
res = a+b

Referencia → suma(10,20,ref R)

void suma(int a,b; ref int r)

Params de entrada → se pasa x ref pero no se puede modificar

↳ + eficiente en algunos casos

(no uso xp)

Params → Mejor más flexibilidad con la cant de parámetros

Método en forma de instrucción

para métodos cortitos.

int suma(int a, int b) => suma(int a,int b) => Console.WriteLine(a+b);

return a+b;