Tema 1:
Introducción a la abstracción

***Objetivos del tema**: El mundo real está compuesto por múltiples objetos: coches, semáforos, personas, lápices, bolígrafos, bancos, teléfonos, etc. Estos objetos forman parte de los problemas del mundo real y desde nuestra más tierna infancia aprendemos a manipularlos para obtener de ellos cierta funcionalidad y resolver problemas que se nos plantean en la vida diaria como, por ejemplo, llamar a un amigo con el que queremos quedar para tomar un café.*

Los programas informáticos pretenden resolver problemas en los que intervienen objetos del mundo real. Para resolver esos problemas será necesario crear una representación software de dichos objetos que capture las propiedades y el comportamiento de los objetos que son relevantes para el problema; esto es, es necesario obtener una abstracción adecuada del objeto del mundo real. El objetivo de este tema es mostrar al alumno como conseguir las especificaciones de dicha representación. También comenzaremos a ver cómo representar las propiedades de dichos objetos en los lenguajes de programación; más adelante en la asignatura veremos cómo representar su comportamiento.

CEU

1. Un mundo de objetos

Desde una edad temprana, aprendemos que el mundo está compuesto por objetos que tienen una serie de propiedades (color, forma, tamaño, textura, peso, etc.) y que podemos obtener una cierta funcionalidad de ellos si los manipulamos del modo adecuado. Los bebés, por ejemplo, aprenden que si mueven un sonajero hará ruido.

La visión del mundo como un conjunto de objetos facilita el manejo de su complejidad. Consideremos una situación que a todos nos ha sucedido en más de una ocasión. Estamos sentados en casa mirando el televisor y recordamos que nuestro programa favorito empezará pronto en un canal distinto. Tomamos el mando a distancia y apretamos un botón para cambiar el televisor al canal adecuado. El mando a distancia es un objeto físico que tiene propiedades, tales como peso y tamaño, y además puede hacer algo: puede enviar mensajes a la televisión. La mayor parte de nosotros no comprendemos cómo lo hace o cómo están codificados los mensajes, pero no es preciso saberlo. Sólo necesita saber cómo se aprietan los botones adecuados; es decir, sólo necesitamos saber manipular el mando adecuadamente, sin importar su funcionamiento interno. Los botones son la interfaz del mando a distancia. Si se conoce la interfaz de un objeto, se puede usar dicho objeto para hacer algo, aunque no se sepa cómo funciona.

La clave para cambiar el canal sin conocer el funcionamiento interno del mando está en disponer de una abstracción adecuada de este objeto. Una abstracción es un modelo que incluye los aspectos más importantes, esenciales o distinguibles de algo mientras que suprime o ignora los detalles menos importantes que no son relevantes para la tarea en curso y que pueden distraernos de ésta. Así, para cambiar el canal de la televisión para nada necesitamos saber que al pulsar el botón una lámpara de infrarrojos emitirá una secuencia de parpadeos que será interpretada por la televisión como una orden para sintonizar un ancho de banda diferente.

Es importante tener en cuenta que los atributos y la funcionalidad de un objeto que son relevantes para un problema dependen de la naturaleza de dicho problema. Así, por ejemplo, si queremos encender un fuego para una barbacoa unas hojas de papel de periódico o unos folios en blanco son un material fácilmente combustible y cualquiera de ellos va a ser igualmente útil para nuestro propósito. Si por el contrario lo que queremos es tomar notas en clases las hojas de papel de periódico no servirán a nuestro propósito, mientras que los folios sí.

Un ejemplo gráfico bastante ilustrativo de lo qué es la abstracción es el cuadro "La traición de las imágenes" del pintor belga René Magritte. En él aparece dibujado una pipa y bajo esta aparece la frase "esto no es una pipa". Efectivamente, el dibujo es una abstracción o representación conceptual de la pipa, y no una pipa. En el cuadro se pierde la tridimensionalidad de la pipa real, sus imperfecciones, su olor etc. pero se ha mantenido sus características fundamentales: contorno,

color, forma etc. En este sentido, el dibujo de la pipa es una buena abstracción, es un buen modelo de la pipa real.

Esta forma de afrontar nuestro complejo mundo puede ser aplicada al diseño y a la programación de software: un paso clave en la resolución de un problema del mundo real es identificar los objetos que componen dicho problema en el mundo real y obtener una abstracción adecuada de estos que sea simple de implementar en un lenguaje de programación.

Un objeto se distingue por disponer de:

- Un **nombre**
- **Propiedades** asociadas al mismo.
- **Funcionalidad** que puede proporcionar.

A menudo cuando empleamos la funcionalidad del objeto se ejecuta un tipo de acción y/o cambia una propiedad del objeto como consecuencia de dicho mensaje. En el ejemplo del mando a distancia, cuando el televisor recibe el mensaje cambio de canal desde el mando, cambia los canales. Del mismo modo, cuando conducimos un coche al pisar el acelerador el coche incrementa su velocidad (cambia una de sus propiedades). Nuevamente, para conducir un coche no son necesarios conocimientos acerca del funcionamiento interno del motor, de cómo el carburador o la inyección incrementa la cantidad de combustible que entra al cilindro para producir una explosión más fuerte que haga mover al émbolo más rápidamente...

La construcción de abstracciones de los objetos del mundo exterior se apoya en muchas ocasiones en cómo han sido diseñados estos. Este diseño ha tratado de convertirlos en cajas negras cuyo contenido desconoce el usuario (circuitos eléctricos, bombilla de infrarrojos, pilas, etc. en el caso del mando a distancia, pistones, cilindros, válvulas, etc. en el caso del motor del coche) pero que ofrecen una interfase simple e intuitiva para el usuario (botones y el pedal del acelerador, respectivamente). A este principio de diseño se denomina **encapsulación**. Esta puede definirse

como la localización física de las propiedades dentro de una sola abstracción de caja negra que oculta su implementación tras un interfaz público.

La encapsulación en software trata de construir módulos que ocultan la complejidad de su implementación y que ofrecen un interfaz simple para que otros programas puedan acceder a su funcionalidad. El desarrollar software siguiendo este esquema permite construir módulos reutilizables en múltiples programas sin necesidad de que sus programadores conozcan los detalles de la implementación de cada uno de estos módulos, al igual que nosotros no necesitamos conocer cómo funciona el mando a distancia de la televisión o el motor de un coche para obtener funcionalidad de ellos.

1.1 Clases y categorías de objetos

Volviendo al ejemplo del mando de la televisión, supongamos que estamos en casa de un amigo y éste tiene un televisor distinto al nuestro y, por tanto, un mando distinto. Supongamos que nunca hemos visto su televisión. A pesar de esto seremos capaces de emplear el mando para cambiar el canal de la televisión, aunque no conozcamos este objeto (mando) en concreto. Esto se debe a nuestra capacidad para generalizar y abstraer las propiedades los objetos del mundo real en clases. Una **clase** es una descripción de un conjunto de objetos que poseen unos mismos atributos y un comportamiento común. Una clase agrupa un conjunto de objetos similares del mundo real. Así, sabemos que un mando de televisión permite cambiar el canal del televisor, y que tendrá una serie de botones que deberemos accionar para conseguir nuestro propósito.

A menudo para resolver un problema del mundo real a través de un programa debemos plasmar en una representación computable una serie de propiedades y comportamiento correspondiente a una clase del mundo real. Para ello se emplean habitualmente los lenguajes de programación.

2. Lenguajes de alto nivel

2.1 ¿Por qué lenguajes de programación?

El ordenador solo entiende el lenguaje máquina escrito en bits, es decir de 0s y 1s. Es muy difícil, algunos dirían que imposible, programar en ese lenguaje, por ello nació el lenguaje ensamblador, un lenguaje de bajo nivel utilizado para representar las instrucciones en lugar de largas cadenas de bits. Esto facilita a los programadores la escritura y lectura del código, aunque todavía requiere un conocimiento detallado del hardware subyacente.

A medida que la tecnología avanzaba y los programas se volvían más complejos, nació la necesidad de lenguajes de programación de más alto nivel. Estos nuevos lenguajes permitieron a los programadores escribir código de manera más sencilla, utilizando palabras clave y estructuras cercanas al lenguaje humano.

2.2 C#: el lenguaje

C# es un lenguaje de programación de gran importancia, destaca por su versatilidad y potencia en el desarrollo de aplicaciones modernas. Diseñado por Microsoft en el año 2000 como parte de la plataforma .NET. Se creo principalmente para la creación de una de aplicaciones Windows y WebApps con ASP.NET. Hoy en día su uso se ha extendido, por ejemplo al desarrollo de aplicaciones móviles (Xamarin) o videojuegos. C# es crucial para el diseño de videojuegos por su integración en motor de juegos Unity, que es uno de los motores más populares y ampliamente utilizados en la industria del desarrollo de videojuegos.

Algunas características importantes de C#:

- **Simple**: C# fue diseñado para ser simple y fácil de aprender, con una sintaxis clara y concisa. Se basa en el lenguaje C, pero elimina o simplifica características complicadas, como la aritmética de punteros.

- **Lenguaje de tipado fuerte:** C# es un lenguaje de programación en el que los tipos de datos de las variables y expresiones están estrictamente definidos. El lenguaje impone reglas estrictas sobre cómo los tipos pueden ser utilizados e interactuar entre sí.

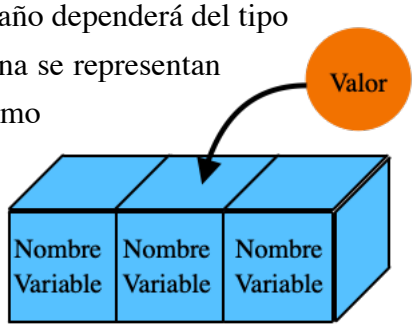
- **Orientado a objetos**: C# es un lenguaje totalmente orientado a objetos. Todo en C#, a excepción de los tipos de datos primitivos (int, char, float, etc.), es un objeto, lo que facilita la organización y reutilización del código.

- **Gestión automática de memoria**: A diferencia de otros lenguajes, incorpora características modernas como la gestión automática de memoria a través del recolector de basura. Esto facilita su uso y reduce los errores, los desarrolladores no necesitan preocuparse por la liberación manual de memoria, lo que simplifica el proceso de desarrollo y reduce la carga cognitiva.

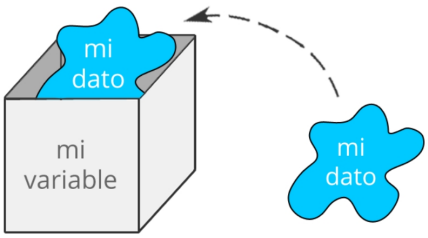
- **Seguro y robusto**: C# incluye una serie de características de seguridad que ayudan a prevenir errores comunes en tiempo de ejecución. Por ejemplo, el manejo de excepciones está integrado en el lenguaje, lo que permite detectar y manejar errores de manera eficiente. Además, el compilador de C# realiza una verificación de tipos estricta y otras comprobaciones que contribuyen a la robustez del código.

3. Variables

Las variables son identificadores utilizados para almacenar datos. Cada variable se asocia con una determinada zona dentro de la memoria RAM del ordenador cuyo tamaño dependerá del tipo de dato que almacene la variable. Todos los datos a nivel de la máquina se representan en una secuencia de bits; sin embargo los lenguajes de alto nivel, como C#, permiten basarse en abstracciones que permiten ignorar los detalles de la representación interna de los datos; esto es, nos permiten olvidarnos de que ese dato es realmente un conjunto de bits y podemos pensar que es un número entero o un número real, por ejemplo.



Las variables pueden contener diferentes valores durante los diferentes pasos de la ejecución de un programa. Podemos pensar en una variable como en una especie de "caja" donde podemos meter cosas diferentes en instantes de tiempo diferentes de la ejecución de un programa (la figura a la derecha de este párrafo trata de ejemplificar ese concepto).



Todas las variables en C# deben definirse antes de su uso. Una variable se define la siguiente forma:

```
| tipo_de_dato  nombre_variable;
```

esta sentencia define la variable nombre_variable que almacenará un dato del tipo tipo_de_dato. El tipo_de_dato nos indica qué tipo elemento vamos a introducir en la “caja”, y cuanto espacio puede llegar a tener el objeto en la “caja”.

Es posible definir varias variables del mismo tipo a la vez:

```
| tipo_de_dato nombre_variable1, nombre_variable2,..., nombre_variableN;
```

Definir una variable simplemente significa que se ha reservado espacio en memoria para dicha variable. Tenemos la caja donde podemos meter algo. Pero todavía no hemos metido nada en la caja. Al definir una variable es posible inicializarla, esto es, darle un valor inicial.

```
| tipo_de_dato nombre_variable = valor;
```

Al inicializar una variable estamos "metiendo" el valor de inicialización en la “caja” correspondiente con la variable. El operador "=" no tiene significado de igualdad matemática, sino que significa "meter lo que hay a la derecha en lo que hay a la izquierda”. A nivel físico dentro del ordenador, esto quiere decir que un conjunto de bits que representan el valor con el que estamos inicializando la variable se va a guardar en las posiciones de memoria RAM asignadas a dicha

variable. Es posible en instantes de tiempo diferentes del programa asignar valores diferentes a una variable; en este caso lo que estamos haciendo sería "meter" algo diferente dentro de la misma caja:

```
| nombre_variable = otro_valor;
```

Una variable es una "única caja"; por tanto, si metemos algo diferente en ella necesariamente tenemos que quitar lo que ya tenía; no es posible representar dos cosas diferentes con los mismos bits de una región de memoria RAM. Si asignamos un nuevo valor a una variable, perdemos su valor antiguo. Si intentamos acceder al valor de la variable antes de inicializarla ésta puede contener un valor cualquiera que podría diferir dependiendo de la plataforma en la que se ejecute el programa, por lo que siempre debemos darle un valor inicial a las variables antes de usarlas y no confiar en la inicialización por defecto del compilador. El nombre de una variable es una forma simbólica de llamar a la dirección de memoria donde comienza el dato que almacena. Los nombres de las variables deben guardar relación con el objeto que representa.

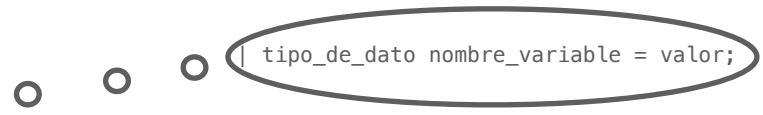
En C# existen una serie de tipos de datos básicos que es posible componer para crear tipos de datos más complejos adaptados a las necesidades particulares de un programa. Empezaremos viendo cuáles son los tipos de datos básicos.

4. Tipos de datos básicos

Hay cuatro categorías básicas de tipos de datos: caracteres, números enteros, números reales y booleans.

4.1 Caracteres (char)

El tipo de datos **char** se emplea para almacenar **caracteres**, o enteros de 8 bits. Los literales de tipo carácter se representan mediante un único carácter encerrado entre comillas simples. Así por ejemplo:




```
| char letra = ‘a’;
```

define una variable de tipo carácter que se llama letra y que contiene el carácter a. El valor en caracteres, como se comentó anteriormente, va siempre entre comillas simples.

Los caracteres en sí son números enteros. Los caracteres se representan en el ordenador mediante el código ASCII, American Standard Code for Information Interchange (Código Estadounidense Estándar para el Intercambio de Información), un código de caracteres basado en el alfabeto latino tal como se usa en inglés moderno y otras lenguas occidentales. Creado en 1963 por el Comité Estadounidense de Estándares (ASA) como una evolución de los conjuntos de códigos utilizados entonces en telegrafía. A continuación mostramos la tabla de caracteres ASCII:

Dec	Hex	Binary	Char	Dec	Hex	Binary	Char	Dec	Hex	Binary	Char	Dec	Hex	Binary	Char
0	0x00	00 00000	NUL	32	0x20	01 00000	SPACE	64	0x40	10 00000	@	96	0x60	11 00000	`
1	0x01	00 00001	SOH	33	0x21	01 00001	!	65	0x41	10 00001	A	97	0x61	11 00001	a
2	0x02	00 00010	STX	34	0x22	01 00010	"	66	0x42	10 00010	B	98	0x62	11 00010	b
3	0x03	00 00011	ETX	35	0x23	01 00011	#	67	0x43	10 00011	C	99	0x63	11 00011	c
4	0x04	00 00100	EOT	36	0x24	01 00100	$	68	0x44	10 00100	D	100	0x64	11 00100	d
5	0x05	00 00101	ENQ	37	0x25	01 00101	%	69	0x45	10 00101	E	101	0x65	11 00101	e
6	0x06	00 00110	ACK	38	0x26	01 00110	&	70	0x46	10 00110	F	102	0x66	11 00110	f
7	0x07	00 00111	BEL	39	0x27	01 00111	'	71	0x47	10 00111	G	103	0x67	11 00111	g
8	0x08	00 01000	BS	40	0x28	01 01000	(	72	0x48	10 01000	H	104	0x68	11 01000	h
9	0x09	00 01001	HT	41	0x29	01 01001	)	73	0x49	10 01001	I	105	0x69	11 01001	i
10	0x0a	00 01010	LF	42	0x2a	01 01010	*	74	0x4a	10 01010	J	106	0x6a	11 01010	j
11	0x0b	00 01011	VT	43	0x2b	01 01011	+	75	0x4b	10 01011	K	107	0x6b	11 01011	k
12	0x0c	00 01100	FF	44	0x2c	01 01100	,	76	0x4c	10 01100	L	108	0x6c	11 01100	l
13	0x0d	00 01101	CR	45	0x2d	01 01101	-	77	0x4d	10 01101	M	109	0x6d	11 01101	m
14	0x0e	00 01110	SO	46	0x2e	01 01110	.	78	0x4e	10 01110	N	110	0x6e	11 01110	n
15	0x0f	00 01111	SI	47	0x2f	01 01111	/	79	0x4f	10 01111	O	111	0x6f	11 01111	o
16	0x10	00 10000	DLE	48	0x30	01 10000	0	80	0x50	10 10000	P	112	0x70	11 10000	p
17	0x11	00 10001	DC1	49	0x31	01 10001	1	81	0x51	10 10001	Q	113	0x71	11 10001	q
18	0x12	00 10010	DC2	50	0x32	01 10010	2	82	0x52	10 10010	R	114	0x72	11 10010	r
19	0x13	00 10011	DC3	51	0x33	01 10011	3	83	0x53	10 10011	S	115	0x73	11 10011	s
20	0x14	00 10100	DC4	52	0x34	01 10100	4	84	0x54	10 10100	T	116	0x74	11 10100	t
21	0x15	00 10101	NAK	53	0x35	01 10101	5	85	0x55	10 10101	U	117	0x75	11 10101	u
22	0x16	00 10110	SYN	54	0x36	01 10110	6	86	0x56	10 10110	V	118	0x76	11 10110	v
23	0x17	00 10111	ETB	55	0x37	01 10111	7	87	0x57	10 10111	W	119	0x77	11 10111	w
24	0x18	00 11000	CAN	56	0x38	01 11000	8	88	0x58	10 11000	X	120	0x78	11 11000	x
25	0x19	00 11001	EM	57	0x39	01 11001	9	89	0x59	10 11001	Y	121	0x79	11 11001	y
26	0x1a	00 11010	SUB	58	0x3a	01 11010	:	90	0x5a	10 11010	Z	122	0x7a	11 11010	z
27	0x1b	00 11011	ESC	59	0x3b	01 11011	;	91	0x5b	10 11011	[	123	0x7b	11 11011	{
28	0x1c	00 11100	FS	60	0x3c	01 11100	<	92	0x5c	10 11100	\	124	0x7c	11 11100	|
29	0x1d	00 11101	GS	61	0x3d	01 11101	=	93	0x5d	10 11101	]	125	0x7d	11 11101	}
30	0x1e	00 11110	RS	62	0x3e	01 11110	>	94	0x5e	10 11110	^	126	0x7e	11 11110	~
31	0x1f	00 11111	US	63	0x3f	01 11111	?	95	0x5f	10 11111	_	127	0x7f	11 11111	DEL

Esta tabla sólo emplea 7 bits para representar cada carácter, mientras que un char tiene 16 bits. Gracias a estos bits extra podemos representar más de 65.000 caracteres, donde entran todos los alfabetos y símbolos, esta extensión se conoce como Unicode.

Hay ciertos caracteres especiales (también se les denomina secuencias de escape) cuya representación textual no coincide con su representación en pantalla (una tabulación, el carácter de nueva línea, etc.), pudiendo incluso no tener ningún tipo de presentación en pantalla sino realizando alguna acción (emitir un sonido, avanzar una página, etc.) o poseyendo un significado especial dentro del lenguaje de programación (el carácter de fin de cadena). Estos caracteres de escape son:

- '\0' carácter de fin de cadena
- '\a' carácter de alarma
- '\b' retroceso
- '\n' nueva linea
- '\r' retorno de carro
- '\t' tabulador horizontal
- '\v' tabulador vertical
- '\\' barra invertida
- '\'' comillas sencillas
- '\"' comillas dobles

4.2 Cadena de caracteres (string)

Para declarar una cadena de caracteres, es decir, una palabra o una frase, se utiliza **string**. Para diferenciar las cadenas de caracteres de los caracteres simples, definidos con char, se utilizan comillas dobles (“”). Un ejemplo de string es:

```
| string saludo = “Hola Mundo”;

| string color = “Amarillo”;
```

Más adelante en la asignatura, veremos posibles operaciones soportadas por la clase string, como la concatenación, comparación y división de la cadena. El tamaño que utiliza string es dinámico y cambia dependiendo del tamaño de la cadena que la variable almacene.

4.3 Valores Lógicos (bool)

Se emplean para representar valores lógicos, esto es, datos que sólo pueden ser ciertos o falsos. El tipo de dato en este caso es *bool*, y solo puede tomar dos valores: “true” (cierto) o “false” (falso). Así por ejemplo:

```
| bool comprobar = true;
```

define una variable lógica que se llama comprobar y que esta inicializada a verdadero. Necesita solo 1 bit (0 o 1), aunque bool ocupa 1 byte (8 bits) por cuestiones de compatibilidad y gestión.

4.4 Números enteros

Los números enteros, como su propio nombre indica, representan datos numéricos que no tienen parte decimal. Todos los números enteros se representan en código binario dentro del ordenador. Existen varias formas de representar un número entero en código binario. Los números enteros sin signo se pueden representar por su binario correspondiente:

25 → 00011001

Mientras que para los números enteros con signo se puede emplear la representación en signo-magnitud (SM): el bit que va más a la izquierda representa el signo (0 si es un número positivo, 1 si es un número negativo) y el resto representan la magnitud.

25 → 0 0011001 -25 → 1 0011001

En C#, los tipos de datos enteros pueden ser **byte**, **char**, **short**, **int** o **long** que almacenan, habitualmente, enteros de 8, 8, 16, 32 o 64 bits, respectivamente.

```
| uint altura, peso;

| long distancia, contabilidad, habitantes;
```

A casi todos los tipos se les pueden aplicar los modificadores signed y unsigned para que sean con signo o sin signo. Los tipos con signo tienen un bit menos de longitud. El tipo char puede corresponder a signed char o unsigned char dependiendo del compilador, en C# pertenece a unsigned. Los rangos y tamaños de cada tipo son:

Palabra tipo C#	Tamaño	Intervalo
sbyte	1 byte con signo (8 bits)	-128 a 127
byte	1 byte sin signo(8 bits)	0 a 255
char	2 byte sin signo(16 bits)	0 a 65.535
short	2 byte con signo (16bits)	-32.768 a 32.767
ushort	2 byte sin signo(16 bits)	0 a 65.535
int	4 byte con signo (32 bits)	-2.147.483.648 a 2.147.483.647
uint	4 byte sin signo(32 bits)	0 a 4.294.967.295
long	8 byte con signo (64 bits)	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
ulong	8 byte sin signo(64 bits)	0 a 18.446.744.073.709.551.615

Los números enteros se pueden representar tanto el formato decimal (455), en binario (para ello se debe anteponer 0b o 0B para indicar que se trata de un numero binario: 0b_0010_1010) y en base 16, conocido como hexadecimal (se antepone un 0x o 0X delante del número: 0xFA81). Así por ejemplo:

```
| short edad = 56; //en decimal

| int altura = 0b110_101; //53 en decimal

| long distancia = 0x773ADA48; //2000345672 en decimal
```

En el ejemplo anterior también se muestra el uso de "_" como un *separador de dígitos*. Puede usar el separador de dígitos con todos los tipos de literales numéricos para mejorar su lectura.

4.5 Números reales

Los números reales se emplean, habitualmente, para representar números que tienen una parte decimal; en ocasiones también se emplean para representan números enteros cuyo tamaño demasiado grande para ser representado por los datos de tipo enteros. Los tipos de datos reales son **double** y **float**. Los rangos y tamaños de cada tipo son:

Palabra tipo C#	Tamaño	Intervalo
float	4 byte con signo (32 bits)	1.5×10^{-45} a 3.4×10^{-38}
double	8 byte sin signo(64 bits)	5.0×10^{-324} a 1.7×10^{-308}

El **double** almacena números de mayor rango y con una mayor precisión que el float. Un ejemplo de uso es:

```
| double pi = 3.141592653;
```

En este ejemplo se da valor a la variable "pi" que es un double. Si al número con decimales no le añadimos ningún sufijo será siempre un double.

El principal beneficio de trabajar con **float** es su tamaño, ocupa menos memoria. Adecuado para aplicaciones donde la precisión no es crítica y el ahorro de memoria es importante. Si un dato con coma flotante lo queremos identificar como float, debe tener un sufijo: f o F. Un ejemplo de uso es:

```
| double altura 1.84F;
```

5. Tipado implicito

El tipado implícito se encarga de asignar directamente un tipo de dato a la variable teniendo en cuenta el valor que tiene asignado. Las variables se declaran con la palabra reservada **var**. Es importante saber que var no es un tipo, sino una manera de declarar variables. Para usarlo, la variable debe ser local y debe estar siempre inicializada. Esto es posible gracias al compilador, que con el dato de la inicialización asigna un tipo a la variable directamente.

```
| var altura = 1.70; //creará un double
```

```
| var nombre = “Patricia"; //creará un string
```