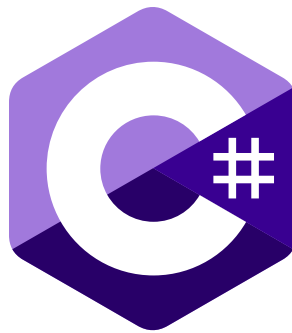




Clase 1: Intro

Bienvenido

En esta asignatura de **Fundamentos de la Programación** — seguiremos programando. Lo único que en vez de utilizar el lenguaje de programación JAVA — utilizaremos otro de los lenguajes más utilizados que es **C#**. Un lenguaje de Microsoft.



Procedimiento asignatura

Como vamos a proceder a lo largo de la asignatura?

Pues exactamente igual que como vimos en la **asignatura de Programación**:

▼ Guía de aprendizaje. Anexos	
	Guia_FPRO_2526.pdf
	Planificacion-Fundamentos_Programacion_T1.pdf
▼ Recursos adicionales	
	Códigos asignatura
	Diario de clase, glosario y ejercicios

Tenemos la guía y la planificación.



IMPORTANTE

Las fechas de la planificación están erróneas. Porque en las fechas ponía que el primer día de clase era la semana pasada, pero las fechas son incorrectas.

Además tenemos todos los contenidos de los que vamos a ir viendo a lo largo de la asignatura.

Hay que tener en cuenta que esta asignatura a diferencia de la de **programación** no es tan grande. Por lo tanto no tenemos tantas unidades. Y dentro de las unidades contenido más reducido.

Recursos adicionales



En recursos adicionales Borja (nuestro profesor) nos ha publicado dos cosas — los códigos de la asignatura y el diario de clases.

▼ Recursos adicionales	
	Códigos asignatura 
	Diario de clase, glosario y ejercicios 

Unidad 1: Introducción

Dentro de la primera unidad tendremos 4 unidades.

En la primera tendremos una pequeña introducción a C#:



▼ Unidad 1. Introducción al desarrollo con C#	
La programación es el proceso de generar software específico utilizando ordenes que son ejecutadas por el ordenador a modo de instrucciones. Estas órdenes son expresadas mediante un lenguaje de programación implementando algoritmos lo más eficientes para que la ejecución sea lo más óptima posible. Sin embargo, programar no es solo tirar líneas de código, sino también comprender el proceso completo de generación de soluciones hasta las pruebas de la solución generada. En esta unidad aprenderemos los conceptos clave para poder aprender a programar, utilizando C# como lenguaje de programación, su estructura interna y los principales elementos del código.	
Temario	
	Introducción al desarrollo con C#
	Estructura básica: operadores, clases, variables y funciones

Veremos también que es una clase... que es un método.. que son los parámetros, los retornos, como podemos escribir una variable, que son los getters y setters,

namespace, funciones que retornen....

Con esto aproximadamente llegaremos hasta primer trimestre. El primer tema es muy sencillito, y en el segundo tema nos extenderemos unas clases.

Actividades:


Actividades
 Lenguajes de programación y entornos de desarrollo
 Resolución de programas utilizando estructuras básicas

Tendremos una serie de actividades — pero no es necesario entregarlas. Cuando haya una actividad entregable nos avisara Borja.

Como hemos dicho, este primer tema es una pequeña introducción y ya instalar y configurar el entorno de desarrollo y empezar a picar código.

Unidad 2: Decisiones y manejo de datos

En la segunda unidad veremos que son esas estructuras de control, como podemos controlar el flujo del programa (no solo de arriba a abajo) sino de arriba a abajo y en algún momento determinado queremos dar una vuelta, y ejecutar lo que esta (por ejemplo 4 líneas por detrás).

▼ Unidad 2. Decisiones y manejo de datos
La programación estructurada es aquella que ejecuta ordenes una detrás de otra, pero no resulta muy útil si no se pueden generar decisiones lógicas en momentos determinados. Para eso, tenemos a nuestra disposición estructuras de control que permiten dirigir el flujo del código según se vaya ejecutando. Además, todas estas decisiones pueden influir en el manejo de datos, necesitando estructuras que permitan aglutinar el conjunto de estos para tratarlos como una sola unidad. En esta unidad aprenderemos a manejar conjuntos de datos además de manejar el flujo de nuestro código de forma correcta.
Temario
 Estructuras de control y estructuras de datos

O en un momento determinado quiero darle una vuelta y ejecutar 20 veces una misma línea de código. O quiero ejecutar 20 veces la línea de código o no dependiendo de lo que vale una variable.

Hablaremos de que son las estructuras de control.

Luego hablaremos de lo que son las **estructuras de datos** — si bien en el primer tema hablaremos de que son las *variables* (que tipo tenemos, como usarlas...) →

aquí lo que hablaremos serán de las estructuras que nos permiten almacenar en un solo sitio una serie de valores.

Es decir, no solo tenemos una variable a la que asignamos un número, sino una variable en la que almacenamos 80 números.

Es lo que se conoce como una **estructura de datos**. Y hablaremos de tres grandes estructuras de datos:

- **Array:** algo que no crece. Como un armario de 4 huecos, en cada hueco puedo guardar algo. Pero no podemos guardar 5. Porque es una estructura de datos estática. Ni la puedo hacer crecer, ni decrecer. Tampoco le podemos quitar un hueco.
- **ArrayList:** Exactamente de la misma forma, queremos tener en una variable guardado 20 valores. Tendremos un armario de 20 huecos, pero si queremos tener una más, le añadimos otro cajón. Podemos añadir o quitar los cajones que queramos. Es dinámico.
- **Diccionarios:** Es lo que se conoce como una lista, pero lo que pasa es que los huecos están estructurados de una forma especial. Tanto en los Arrays como los ArrayList, si quiero pillar el primer hueco es el **0** — la posición 0 es la primera posición en programación, no el 1.

Pues en un ArrayList si queremos acceder al hueco (posición/índice) 4, haríamos:

- 0
- 1
- 2
- 3
- **4**

Y pilláramos el contenido del 4.

Sin embargo, en un **Diccionario** (la estructura de datos) las posiciones no tienen importancia. Lo que importan son los **datos**.

Los valores están asignados por claves. **key/value** pairs.

```
nombre: "Adrian";
```

La clave sería `nombre` y el valor `"Adrian"`.

No tenemos que recorrer y buscar, simplemente conocer la clave.



Hay un tipo de formato llamado JSON (JavaScript-Object-Notation)

Que es muy muy útil en programación. Porque es una forma muy efectiva de transmitir datos de un origen a destino que me permite jerarquizar la información y acceder a ella muy fácil.

Se vería algo así:

```
{
  "nombre": "Ejemplo",
  "edad": 30,
  "ciudad": "Barcelona",
  "esActivo": true,
  "hobbies": ["leer", "viajar", "programar"]
}
```

Es igual que un ***diccionario***.

Unidad 3: Conceptos avanzados

Aquí estaríamos hablando ya de la ***programación orientada a objetos (POO)*** — tendremos una clase que crea el molde de lo que luego es el objeto (la instancia) y hago el objeto real.

▼ Unidad 3. Conceptos avanzados

El gran potencial de la mayoría de los lenguajes de programación es la capacidad que tienen de generar elementos cuyas características y funcionalidades se pueden tratar de forma independiente, así como relacionarlos entre ellos o incluso generar una línea de dependencia entre los diferentes elementos generados. Esto se consigue con la programación orientada a objetos, el principal paradigma de programación. En esta unidad aprenderemos a crear y manejar objetos, así como crear herencia para gestionar dependencias y controlar los posibles fallos que puedan existir en tiempo de ejecución dentro del código de programación.

Temario

🔗 POO y herencia

🔗 Excepciones y librerías adicionales

Tenemos un objeto que es **alimento** — y luego tenemos otro objeto que es **galleta, carne, leche, zumo....** El molde es la clase del objeto, y el objeto es la realidad de la clase.

Hablaremos también sobre la **herencia** — que la propia palabra lo dice — tenemos una clase que es **alimento**, pero debajo de alimento queremos dos clases que hereden y cojan sus características y comportamiento.

También veremos librerías adicionales. Como el JDK que vimos con Java. Que nos da un montón de clases, métodos, estructuras de datos...

Pero si queremos cosas adicionales, podemos pillar librerías externas y descargarnos el fichero, y meterlo dentro de nuestro proyecto.

A partir de ahí tenemos acceso al contenido de las librerías adicionales.

En el caso de **C#**, nos tendremos que descargar algo llamado **.net** — que es el entorno de desarrollo. Pero además, podemos descargarnos un gestor de paquetes e instalar librerías.

También veremos excepciones, errores que no son capaces de compilar y no nos deja ejecutar, pero también veremos errores que podemos lanzar nosotros mismos. Son situaciones en las que le damos al **play** (compilar) → sin embargo hay un error que el compilador no detecta. O que durante la ejecución el programa puede llegar a fallar.

Por ejemplo, que ocurre si dividimos **8 / 0** → esto no se puede hacer. Es infinito, indefinido. Que pasa, que un programa que realiza divisiones (por ejemplo), si mete **8** y **4** lo divide y son **2**. Pero que pasa si mete **0** como segundo operando? Pues ahí habrá un error en tiempo de ejecución.

El programa se para, pero yo como programador tengo que prever esos casos. Eso se conoce como **excepciones**. No podemos dejarlo al libre albedrío.

Por ejemplo imagínate que tenemos que trabajar con un fichero, y el programa no encuentra el fichero. Eso va a ser un error mientras el programa se está ejecutando. O puede que no tengamos permisos de guardar. El código está bien, pero hay un tercero (un error externo) que tenemos que prever. O una conexión a base de datos fallida. Esto es una **excepción**.



En Java: A veces ni nos deja compilar si ve que hay posibilidad que exista una excepción.

Por esto JAVA, al ser más restrictivo se suele utilizar como primer lenguaje.

Pero hay veces que en **C#**, Python etc... Podemos meter la pata si no compilamos ciertas situaciones.

Unidad 4: Transacción de datos: base de datos y ficheros

Aquí en la **unidad 4** — haremos trabajos con ficheros y bases de datos.

▼ Unidad 4. Transacción de datos: base de datos y ficheros

Uno de los puntos clave dentro de todo programa es la consistencia de los datos. Por ello es clave entender cuáles son todas las posibilidades que nos ofrece C# para trabajar con ellos, destacando entre ellas el tratamiento de ficheros (tanto para la lectura como la escritura), la gestión de bases de datos relacionales y sus operaciones CRUD (create – read – update – delete) y por último la conexión a servicios externos para la obtención o tratamiento de datos, lo que se conoce como llamadas a APIs. En esta unidad aprenderemos la gestión de todos estos servicios, utilizándolos para incorporar y gestionar los datos dentro de nuestros programas.

Temario

🔗 Trabajo con ficheros

📄 Bases de datos

Vamos a abrir un **sublime** para verlo mejor.

Cuando nosotros trabajamos con ficheros, en programación se puede trabajar con cualquier tipo de fichero.

```
.docx  
.pdf  
.xls  
.html
```

Lo que ocurre es que para cada una de estas funcionalidades, nativamente no podemos tratarlos. Pero si tenemos un código de programación y que estos datos que hay en mi código pasen al **pdf** de turno, sabes lo que necesitaríamos?

```
.docx  
.pdf    CÓDIGO -> DATOS -> XLS  
.xls  
.html
```

Entre el código y el resultado final, cuando compilemos, queremos que los datos se queden en un excel. O un pdf o lo que sea.

LO PODREMOS HACER SOLO SI TENEMOS LIBRERIAS.

```
.docx  
.pdf    CÓDIGO -> DATOS -> PLAY -> (SOLO SI TENGO LIBRERIAS) -> XLS  
.xls  
.html
```

Porque estos ficheros tienen un formato muy específico. El fomato pdf, el docx, el pptx ...

Borja nos dejara información de que librerías tenemos que utilizar, y que es lo que tenemos que hacer. Pero cuando nosotros trabajemos con ficheros lo normal es trabajar con:

Ficheros con los que solemos trabajar:

1) TXT -> texto plano

Que el sistema procesa cada letra una a una. Y cada una tiene un valor como **byte** en realidad. Esto se refiere al código **ASCII**:

Caracteres ASCII de control				Caracteres ASCII imprimibles				ASCII extendido								
00	NUL	(carácter nulo)	32	espacio	64	@	96	.	128	C	160	á	192	Ł	224	Ó
01	SOH	(inicio encabezado)	33	!	65	A	97	a	129	ü	161	â	193	ł	225	ô
02	STX	(inicio texto)	34	"	66	B	98	b	130	é	162	ã	194	Ł	226	Ô
03	ETX	(fin de texto)	35	#	67	C	99	c	131	à	163	ä	195	ł	227	ö
04	EOT	(fin transmisión)	36	\$	68	D	100	d	132	á	164	å	196	—	228	õ
05	ENQ	(consulta)	37	%	69	E	101	e	133	ä	165	ä	197	+	229	ö
06	ACK	(reconocimiento)	38	&	70	F	102	f	134	å	166	å	198	ä	230	µ
07	BEL	(timbre)	39	'	71	G	103	g	135	ç	167	å	199	Ä	231	þ
08	BS	(retroceso)	40	(72	H	104	h	136	ê	168	æ	200	Ė	232	ð
09	HT	(tab horizontal)	41)	73	I	105	i	137	ë	169	ø	201	Į	233	ŭ
10	LF	(nueva línea)	42	*	74	J	106	j	138	ê	170	÷	202	±	234	Ů
11	VT	(tab vertical)	43	+	75	K	107	k	139	í	171	¼	203	™	235	Ű
12	FF	(nueva página)	44	,	76	L	108	l	140	î	172	½	204	ƒ	236	Ý
13	CR	(retorno de carro)	45	-	77	M	109	m	141	ï	173	¾	205	—	237	Ÿ
14	SO	(desplaza a fuera)	46	.	78	N	110	n	142	À	174	»	206	ƒ	238	—
15	SI	(desplaza adentro)	47	/	79	O	111	o	143	Á	175	«	207	u	239	.
16	DLE	(esc.vínculo datos)	48	0	80	P	112	p	144	Ê	176	—	208	ø	240	=
17	DC1	(control diap. 1)	49	1	81	Q	113	q	145	æ	177	—	209	Đ	241	±
18	DC2	(control diap. 2)	50	2	82	R	114	r	146	Æ	178	—	210	Ė	242	—
19	DC3	(control diap. 3)	51	3	83	S	115	s	147	ö	179	—	211	Ė	243	—
20	DC4	(control diap. 4)	52	4	84	T	116	t	148	ó	180	—	212	Ė	244	—
21	NAK	(conf. negativa)	53	5	85	U	117	u	149	ô	181	À	213	ı	245	\$
22	SYN	(inactividad sinc)	54	6	86	V	118	v	150	û	182	Á	214	ı	246	±
23	ETB	(fin bloque trans)	55	7	87	W	119	w	151	ü	183	Â	215	ı	247	±
24	CAN	(cancelar)	56	8	88	X	120	x	152	ý	184	Ë	216	ı	248	*
25	EM	(fin del medio)	57	9	89	Y	121	y	153	Û	185	Ė	217	ı	249	—
26	SUB	(sustitución)	58	:	90	Z	122	z	154	Ü	186	—	218	ı	250	.
27	ESC	(escape)	59	;	91	[123	{	155	ø	187	—	219	ı	251	.
28	FS	(sep. archivos)	60	<	92	\	124		156	ƒ	188	—	220	ı	252	*
29	GS	(sep. grupos)	61	=	93]	125	}	157	ø	189	—	221	ı	253	*
30	RS	(sep. registros)	62	>	94	^	126	~	158	×	190	—	222	ı	254	■
31	US	(sep. unidades)	63	?	95	_			159	f	191	—	223	ı	255	■
127	DEL	(suprimir)														

Cuando hablamos de **flujos de datos** — estamos hablando de que si queremos trabajar con un flujo de datos entrante o saliente → lo que necesitamos son **librerías si queremos trabajar con algo que NO sea texto plano.**

Ficheros con los que solemos trabajar:

1) TXT -> texto plano - INPUT (lectura) / OUTPUT (escritura)

Además, del texto plano — sin tener que instalar librerías adicionales, podemos realizar un flujo de datos de **objetos**. De la misma forma.

Ficheros con los que solemos trabajar:

1) TXT -> texto plano - INPUT (lectura) / OUTPUT (escritura)

2) OBJ -> objetos - INPUT (lectura) / OUTPUT (escritura)

Solo que ahora estaremos hablando de un **.obj**

Cuando hablamos de objetos, tendríamos por ejemplo una instancia:

2) OBJ -> objetos - INPUT (lectura) / OUTPUT (escritura) - .obj - new Alimento("Carne", 345, 45kg, 300)

Yo esta información, es posible que quiera **exportarla a un fichero**.

Ya no lo vamos a ver como letras, o cadenas de caracteres, ni si quiera si guardamos un número. Lo vamos a guardar **como lo que es — un objeto.ue es — un objeto**

Cuando nosotros lo leamos, lo leeremos como esa caja y lo vamos a ir pudiendo extraer. El nombre, el precio, el peso, las calorías...

Y también hablaremos de un tipo de **fichero especial** — que se llama **JSON** (JavaScript Object Notation).

Es un tipo de fichero muy especial. Porque en teoria es un texto plano, lo que ocurre es que tiene una forma un poco rara.

Ejemplo de JSON:

```
{
  "nombre": "adrian",
  "apellido": "thoenig",
  "edad": 21,
  "amigos": {
    "amigoUno": "Jana",
    "amigoDos": "Nuran",
    "amigoTres": "Marco"
  },
  "esMayorDeEdad": true
}
```

Tendremos este fichero **JSON** — y dentro de el podemos preguntar por una **clave**, que de hecho todo lo que esta en **rojo** son **claves**, y lo que esta en **azúl** son los **valores**:

```
{
  "nombre": "adrian",
  "apellido": "thoenig",
  "edad": 21,
  "amigos": {
    "amigoUno": "Jana",
    "amigoDos": "Nuran",
    "amigoTres": "Marco"
  },
  "esMayorDeEdad": true
}
```

Por ejemplo, si queremos saber el **nombre**, accederiamos a la clave de **nombre** directamente, y como resultado nos daría **adrian**.

Esto sera muy utilizado sobretodo a la hora de hacer **lecturas** donde leeremos un `archivo.json` → y será muy utilizado con el protocolo **HTTP**.



JSON (JavaScript Object Notation) **sirve principalmente para que las máquinas o sistemas se comuniquen entre sí**, intercambiando datos de manera que ambos se entiendan fácilmente.

Por ejemplo:

- Cuando tu navegador pide información a un servidor (como los productos de una tienda o los mensajes de un chat), esa información suele venir en **formato JSON**.
- JSON es **ligero, fácil de leer por los humanos y muy fácil de interpretar por las máquinas**.
- Se usa muchísimo en APIs, bases de datos, y en la comunicación entre el **frontend** (lo que ve el usuario) y el **backend** (lógica de servidor).

Vamos a ver un ejemplo de esto.

Aquí tenemos un *dummy* (ficticio) JSON. Que nos servira de ejemplo:

```
fetch('https://dummyjson.com/products')  
  .then(res => res.json())  
  .then(console.log);
```

Si abrimos el enlace de hecho tenemos un endpoint:

```
{
  "products": [
    {
      "id": 1,
      "title": "Essence Mascara Lash Princess",
      "description": "The Essence Mascara Lash Princess is a popular mascara known for its volumizing and lengthening effects. Achieve dramatic lashes with this long-lasting and cruelty-free formula.",
      "category": "beauty",
      "price": 9.99,
      "discountPercentage": 10.48,
      "rating": 2.56,
      "stock": 99,
      "tags": [
        "beauty",
        "mascara"
      ],
      "brand": "Essence",
      "sku": "BEA-ESS-ESS-001",
      "weight": 4,
      "dimensions": {
        "width": 15.14,
        "height": 13.08,
        "depth": 22.99
      },
      "warrantyInformation": "1 week warranty",
      "shippingInformation": "Ship in 1-5 business days",
      "availabilityStatus": "In Stock",
      "reviews": [
        {
          "rating": 3,
          "comment": "Should not recommend!",
          "date": "2025-04-30T09:41:02.053Z",
          "reviewerName": "Eleanor Collins",
          "reviewerEmail": "eleanor.collins@k.dummyjson.com"
        },
        {
          "rating": 4,
          "comment": "Very satisfied!",
          "date": "2025-04-30T09:41:02.053Z",
          "reviewerName": "Lucas Gordon",
          "reviewerEmail": "lucas.gordon@k.dummyjson.com"
        },
        {
          "rating": 5,

```

Si por ejemplo pedimos el `title` , nos da esto:

```
"id": 1,
"title": "Essence Mascara Lash Princess",
"description": "The Essence Mascara Lash Princess is a popular mascara known for its volumizing and lengthening effects. Achieve dramatic lashes with this long-lasting and cruelty-free formula.",
"category": "beauty",
```

Es decir, veremos como leer un archivo **JSON**, pero no con un fichero que esta en local, sino en una URL (conocido como un endpoint).

Así que estos son los **tres tipos de ficheros con los que trabajaremos**:

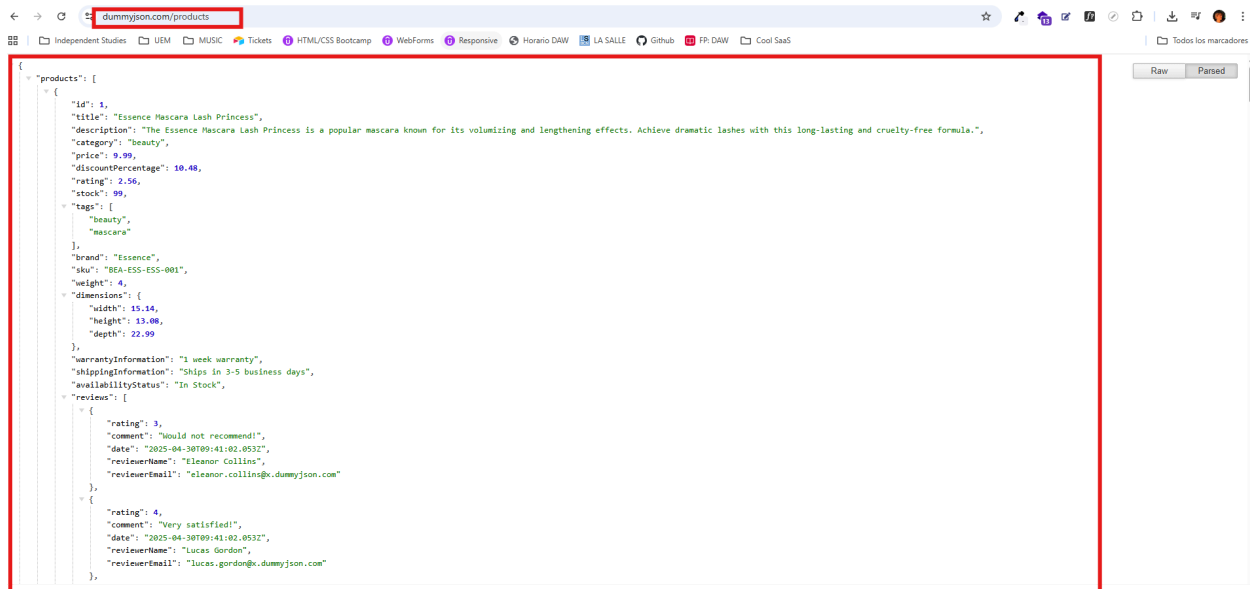
```
Ficheros con los que solemos trabajar:

1) TXT -> texto plano - INPUT (lectura) / OUTPUT (escritura) - .txt
2) OBJ -> objetos - INPUT (lectura) / OUTPUT (escritura) - .obj - new Alimento("Carne", 345, 45kg, 300)
3) JSON -> javascript object notation - INPUT(lectura desde endpoint) / OUTPUT(escritura) - .json

Ejemplo de JSON:

{
  "nombre": "adrian",
  "apellido": "thoenig",
  "edad": 21,
  "amigos": {
    "amigoUno": "Jana",
    "amigoDos": "Nuran",
    "amigoTres": "Marco"
  },
  "esMayorDeEdad": true
}
```

Esto que hemos visto por aquí cuando hemos entrado a la URL de la API del JSON ficticio:



Es bastante importante.

Porque es uno de los *puntos clave en programación*.

También lo tocaremos en la asignatura de **Programación** (con JAVA) — porque tenemos que tener esto al dedillo. Si alguien llega a una empresa y no sabe tratar con un JSON desde el lenguaje de programación pedido, estamos en la m****.

Bases de datos:

Por último, en esta unidad hablaremos de las **Bases de Datos**. Cuando hablemos de bases de datos, hablaremos de una forma de trabajar con ellas que es la más típica de todas.

Lo que pasa es que vamos a mezclar algunas cositas.

Cuando hablamos de bases de datos (en fundamentos de programación), trabajaremos con un motor en concreto que es el **MySQL**.

Bases de datos existen mogollones, lo que pasa es que se pueden diferenciar en dos categorías principales.

- **SQL** (relacionales) → Se basan en sentencias *query*.

```
SELECT * FROM clientes WHERE apellido = "Thoenig";
```

Esto se organiza por **tablas**, y estas tablas tienen **relaciones**. Que conectan una tabla con otra.

Por ejemplo un **usuario** se relaciona con un **departamento** y en un departamento hay muchos usuarios.

Los ejemplos más típicos de este tipo de bases de datos, son los **MySQL**, **PostgreSQL**, **Oracle**, **SQLite...**

Nosotros trabajaremos con **MySQL**.

- **NoSQL**: Ya no tenemos queries, porque ya no hay tablas. Lo que hay aquí son documentos, que están compuestos por algo muy parecido a **JSON**.

Tenemos colecciones, como por ejemplo *Películas*, y tenemos película1, película2....

Así se vería una colección NoSQL:

```
{
  "titulo": "The Matrix",
  "año": 1999,
  "directores": ["Lana Wachowski", "Lilly Wachowski"],
  "reparto": [
    { "nombre": "Keanu Reeves", "personaje": "Neo" },
    { "nombre": "Laurence Fishburne", "personaje": "Morpheus" }
  ],
  "generos": ["Acción", "Ciencia ficción"]
}
```

Y lo que tendríamos serían métodos:

- `db.coleccion.find()`

Cuando nos queremos conectar desde un JAVA, o un Python, PHP, o el que sea... NO lo podemos hacer nativamente, necesitamos si o si una **librería**. Si queremos ejecutar cosas relacionadas con las bases de datos, necesitamos las librerías.

Nosotros vamos a tener nuestro **CLIENTE** en **C#** → y nos queremos conectar con una base de datos, será poner las librerías correspondientes y hacer el acceso a la base de datos.

Lo que pasa, es que si nos conectamos al JSON a través de un HTTP, haremos un **esquema de conexión real de Base de Datos**.

Lo que quiere decir es que cuando yo me conecto a una base de datos directamente, lo estoy haciendo **de una forma insegura**.

Pues que alguien entre medias, de esta comunicación, puede hacer un ataque MITM (*Man In the Middle*) → y captura todas las credenciales de la base de datos, y los datos han quedado expuestos.

CLIENTE (C#) ----- ATACANTE ----- BASE DE DATOS

Y ahí ya puede hacer lo que quiera.

Que es lo que realmente se hace?

Pues lo que hemos hablado antes, se crea un **servidor intermedio**, un **SRV WEB** desde donde el cliente le haremos una **petición HTTP(s)** y hacemos la petición a través de Internet.

CLIENTE (C#) → Petición HTTP → SRV WEB → Base de Datos

Lo que está ocurriendo es que desde el **CLIENTE (C#)** nos hemos conectado al servidor a través de una petición, el servidor evalúa lo que le estamos pidiendo en la petición.

Dependiendo de cómo este programado llevará a un sitio u a otro.

La base de datos le **responde** al Servidor Web, construye todo y lo devuelve a nuestro cliente.

El servidor web lo contesta con un **JSON**.

¿Por qué crees que esta es la mejor arquitectura / manera para conectar con una base de datos?

1. **Seguridad:** Al no hacer un contacto *directo* con la base de datos, evitamos que los datos sean captados por un tercero (*MITM attack*) esto mete una capa extra de seguridad. Desde el servidor podemos validar la petición, si el usuario está autenticado/autorizado, qué roles tiene, etc... Aquí metemos un filtro.

2. **Disponibilidad:** Si hacen un ataque al servidor, lo tiran todo (en caso de que hagamos contacto directo) → nos tiran el servidor web y la base de datos. Sin embargo, si tenemos la infraestructura de **CLIENTE → HTTP(Petición) → SRV WEB → Base de Datos** añadimos algo llamado **disponibilidad** que es que si tiran el servidor, podemos levantar un servidor **secundario** que es exactamente el mismo que el primario pero esta re-equipado. Si cortan una carretera, te desvío por otra carretera.



Ser capaces en Informática de garantizar la disponibilidad de los datos, es ser capaces de que la empresa va a seguir ganando pasta. Porque si tiran una ficha y perdemos todo, la hemos cagado.

Si que es verdad que no entra en nuestro curro como programadores tocar esto, pero si que tenemos que ser conscientes de la existencia de estos conceptos.

Esto sera lo último que trabajaremos en la asignatura de **Fundamentos de Programación OL** — porque ya habremos visto como trabajar con el lenguaje, con los datos, POO, ficheros, bases de datos..

Lo último que haremos sera un **CRUD** (Create-Read-Update-Delete) → todas las opciones que se pueden hacer en una base de datos para directamente trabajar con ella.

C#

C# es un lenguaje propietario de **Microsoft** — y nos centraremos sobretodo en Aplicaciones de Consola, pero se puede utilizar para muchisimas más cosas.

Podemos hacer una página web, una aplicación móvil, una de Servidor e incluso videojuegos en Unity.

.NET

C# está basado en **.NET** → es la plataforma de desarrollo de Microsoft. **C#** es el lenguaje, pero **.NET** es la infraestructura completa para programar.

Lo que necesitamos si o si para programar en **C#**, es descargar **.NET**

IDE → Vs Code

Utilizaremos **VSCode** para desarrollar en C#.

Nuestro primer Hola Mundo

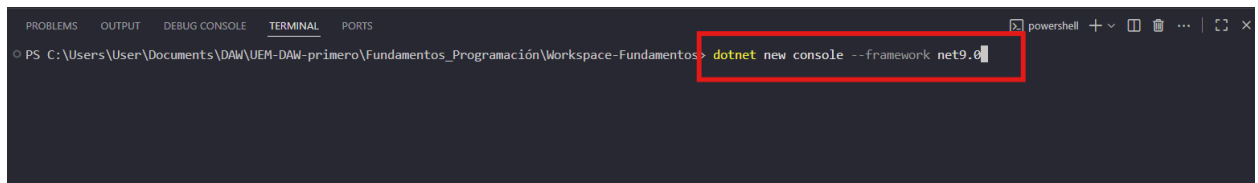
Vamos a crear un **Hola Mundo** en C#:

Para ello tenemos que instanciar una **terminal** en la ruta de nuestro proyecto.

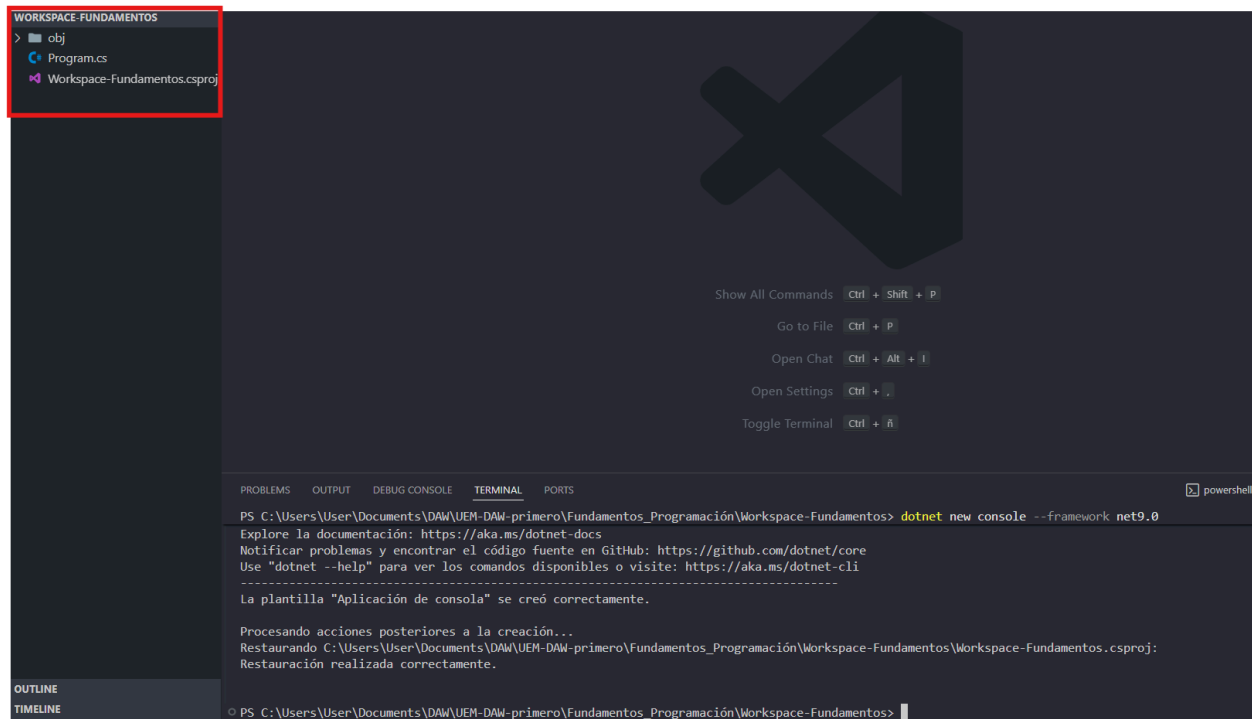


Y introducir el siguiente comando:

```
dotnet new console --framework net9.0
```



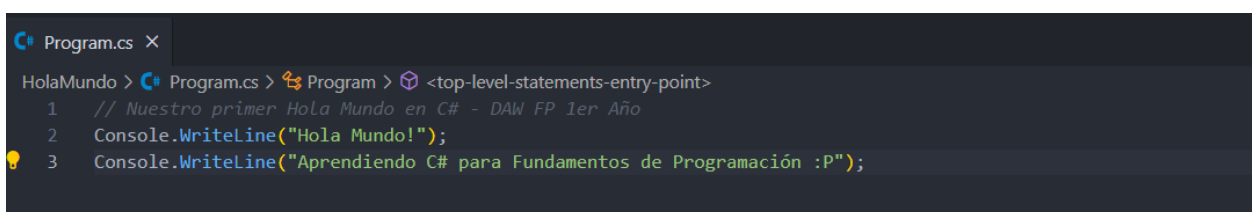
Una vez lo corremos habra creado los archivos iniciales del proyecto.



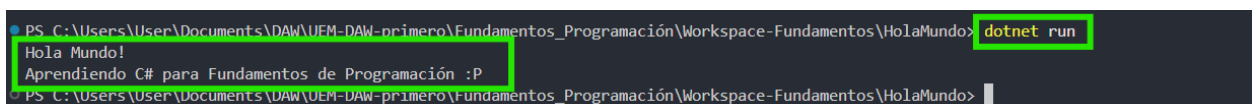
Y dentro de `Program.cs` podemos poner la instrucción para imprimir en pantalla el Hola Mundo:

```
Console.WriteLine("Hola Mundo!");
```

Quedaría así:



Ahora solo faltaria desde la **terminal** ejecutar `dotnet run` :



Y ahí tendríamos nuestro **Hola Mundo** en C#.