Introducción a los lenguajes de marcas

1. Ordenadores y almacenamiento de la información

¿Cómo hace un ordenador para almacenar la información? La respuesta a esta pregunta depende sobre todo del tipo de información que se quiere almacenar. Si queremos almacenar unos apuntes o contenido como el que estamos leyendo, nos puede bastar con un simple archivo de texto, y en ese caso lo que almacenamos es información textual, es decir, información que se almacena guardando una tras otra las diferentes letras y símbolos del documento en cuestión. Pero si pretendemos almacenar una canción, o una fotografía, el modo en que se almacenan los distintos sonidos, o colores de la imagen, puede variar. Los primeros se denominan **archivos de texto**, y los segundos (archivos de sonido, imágenes, etc) se denominan **archivos binarios**.

1.1. Sistemas de codificación

Con los archivos de texto existe un problema de inicio: el idioma o región origen del documento. Un texto escrito en castellano lleva letras acentuadas y eñes, por ejemplo, mientras que un texto escrito en ruso contiene letras del alfabeto cirílico, inexistentes en el alfabeto castellano. ¿Cómo hacer que un archivo de texto almacenado en un idioma concreto pueda abrirse y leerse en una región o sistema con otro idioma asociado? Para solventar este problema se idearon distintos **sistemas de codificación**.

A finales de los años 70 se ideó un sistema de codificación llamado **ASCII** (*American Standard Code for Information Interchange*). Inicialmente los símbolos se codificaban utilizando 7 bits, lo que permitía codificar 128 símbolos. En esta primera aproximación se incluyeron únicamente los símbolos del alfabeto inglés (letras mayúsculas y minúsculas sin acentuar, dígitos del 0 al 9), y algunos símbolos adicionales (signos de puntuación, etc.). Sin embargo, no era suficiente para almacenar otros símbolos existentes en otros alfabetos, como los acentos latinos. Se ideó entonces el sistema **ASCII extendido**, que añadía 1 bit más (1 byte en total) para poder así codificar 128 + 128 = 256 símbolos diferentes. Esto dio cabida a las vocales acentuadas, eñes y otros elementos con diéresis, etc. Podéis consultar aquí un listado completo del sistema ASCII extendido, con el código numérico asociado a cada símbolo.

Aún así, con el sistema ASCII extendido seguían quedando fuera numerosos símbolos, correspondientes a alfabetos más dispares, como el cirílico o el japonés. Era necesario establecer sistemas de codificación más globales. La **ISO** (*International Organization for Standardization*) ideó un sistema para normalizar estas tablas de códigos mediante diferentes normas. Por ejemplo:

- La norma **ISO-8859-1** abarca los caracteres propios de Europa Occidental, incluyendo vocales acentuadas, y símbolos como la eñe.
- La norma **ISO-8859-2** comprende los caracteres propios de Europa Central y Oriental, incluyendo símbolos propios de lenguas eslavas como č.
- ... etc.

El problema de estas agrupaciones es que, a la hora de editar un documento de texto, debemos especificar en cuál de estos sistemas de codificación está escrito, para que el programa que lo abra sepa interpretarlo. El paso final a este proceso se ha dado con los sistemas **Unicode**. Estos sistemas han conseguido recopilar los caracteres de todas las lenguas del planeta, pero para ello se necesita más de un byte para representar cada símbolo. Los 128 primeros elementos corresponden al sistema ASCII original, para mantener la compatibilidad, y los 128 siguientes corresponden al sistema europeo ISO-8859-1. A partir de ahí, se añade un segundo byte para lenguas cirílicas, griegas, etc, o un tercer byte para chino o japonés, o un cuarto byte para símbolos adicionales (matemáticos, lenguas muertas...). Por lo tanto, la codificación Unicode puede llegar a ocupar hasta 4 bytes, dependiendo del alfabeto que se quiera utilizar. A partir de aquí, existen distintas versiones del sistema Unicode **UTF** (*Unicode Transformation Format*):

- **UTF-8**: utiliza 8 bits (1 byte) para ASCII e ISO-8859-1, 2 bytes para cirílico, griego, etc, 3 bytes para chino o japonés y 4 bytes para símbolos especiales, tal y como se ha explicado antes.
- **UTF-16**: utiliza 2 bytes para los dos primeros grupos anteriores (ASCII, ISO-8859-1, cirílico, griego, etc), 3 bytes para chino o japonés y 4 para el resto
- **UTF-32**: utiliza 4 bytes para todos los sistemas. Es más sencillo de gestionar, aunque ocupa más espacio, por lo que apenas se utiliza.

1.2. Incompatibilidades

En los **archivos binarios** la información se almacena en una secuencia de bits (ceros y unos) agrupados en grupos de 8, llamados *bytes*. Esta es la forma en que un ordenador puede entender la información almacenada para, después, poder reproducir un archivo de audio o mostrar una imagen. Sin embargo, si el programa que utilizamos para abrir o reproducir ese archivo es distinto al programa que se utilizó para generarlo, puede haber un problema de incompatibilidad. Es el gran inconveniente que presentan los archivos binarios: que son dependientes del software y el sistema que se utilizó para generarlos.

Sin embargo, los archivos de texto en principio no presentan estos problemas. Una carta guardada con el bloc de notas de Windows, por ejemplo, la podemos abrir después en un editor de texto de Mac, o de Linux, con el mismo formato o apariencia. Pero comenzamos a tener problemas cuando queremos añadir "algo más" a estos archivos, como por ejemplo información de formato: tipos de letras, tamaños, colores. Esto no se puede representar con un texto, y en este caso, hasta hace no mucho, obligaba a que los archivos de texto se tuvieran que guardar también en formato binario para poder almacenar, junto con el texto, la información del formato. Esto traía consigo los comentados problemas de compatibilidad, y es ahí donde los lenguajes de marcas juegan un papel importante.

2. ¿Qué son los lenguajes de marcas?

Cuando queremos almacenar algún tipo de información, como un documento de trabajo, un manual de usuario, un listado de libros o un artículo, podemos hacerlo de distintas formas. Por ejemplo, podemos emplear un editor de textos avanzado como Microsoft Office Word, o LibreOffice, aunque en este caso tendríamos el inconveniente de que no podríamos consultar el documento sin tener el software instalado, y al intentar abrirlo con otro software diferente se perdería la compatibilidad y se vería diferente, o directamente no podríamos abrir el documento. También podríamos utilizar un archivo de texto plano y una herramienta

sencilla como el bloc de notas, pero entonces nos costaría más identificar o localizar las partes del documento.

Los **lenguajes de marcas** nos permiten, por un lado, poder añadir anotaciones en un documento de texto que hagan referencia a cuestiones como el formato del documento (tipos de letra, tamaños, colores), y por otra parte, estructurar la información de un documento añadiendo una serie de marcas, o etiquetas, que indican qué tipo de información estamos almacenando en cada parte.

2.1. Ejemplos iniciales: TeX y JSON

El formato de etiquetado **TeX** fue ideado en la década de los 70 para producir documentos científicos que, almacenándose en modo textual, fueran compatibles en distintos tipos de sistemas, incluyendo toda la nomenclatura que estos textos pueden contener (fórmulas matemáticas, expresiones, etc.). Para ello, a la información textual se le añadieron ciertas marcas que la complementaban, e informaban al programa encargado de abrir el documento de cómo tenía que mostrarse la información. Aquí vemos un ejemplo breve de este tipo de documentos.

```
\documentclass[12pt]{article}
\usepackage{amsmath}
\title{\Ejemplo}
\begin{document}
Este es el texto ejemplo de \LaTeX{}
Con datos en \emph{cursiva} o \textbf{negrita}.
Ejemplo de f\'ormula
\begin{align}
E &= mc^2
\end{align}
\end{document}
```

Como podemos ver, se especifican mediante marcas cosas como el tamaño de letra, encabezados o títulos, acentos, cursivas...

Por otra parte, el formato **JSON** (*JavaScript Object Notation*) fue una derivación del lenguaje de programación JavaScript para idear una forma textual de representar objetos de una aplicación. Supongamos que queremos almacenar información de una lista de contactos, tales como nombres, direcciones y teléfonos. En formato JSON, podríamos almacenarlo así:

```
[
    {
        "nombre": "Juan Pérez",
        "direccion:": "C/Ávila, 12",
        "telefono": "611223344"
    },
    {
        "nombre": "Elena García",
        "direccion": "Avda. de las Naciones, 25",
        "telefono": "600998877"
    },
    ...
]
```

Como podemos intuir, esta información se puede trasladar cualquier sistema, y leer fácilmente desde distintos programas.

2.2. Origen y evolución de HTML. El lenguaje SGML y derivados

Ya en los años 60 del siglo pasado las empresas que se dedicaban a publicar o gestionar documentación electrónica se encontraban con el inconveniente de la falta de compatibilidad: documentos creados con una aplicación no podían ser abiertos o editados con otra aplicación diferente. Esto dificultaba que se pudieran compartir documentos.

Ante esta problemática, la empresa IBM desarrolló un lenguaje de marcas primitivo llamado **GML** (*Generalized Markup Language*), que empleaba marcas precedidas del símbolo del doble punto (😃 para estructurar el documento de forma genérica: listas, tablas y párrafos. Posteriormente, a mediados de los 80, ISO utilizó esta base para definir **SGML** (*Standard Generalized Markup Language*), que básicamente es un estándar para definir lenguajes de marcado para documentos. Todos los lenguajes de marcas actuales descienden de él.

Ya en los 90, se define el **HTML** (*HyperText Markup Language*) a partir de la sintaxis de SGML. El **Consorcio World Wide Web** (*W3C*) regula las recomendaciones y versiones normalizadas de este lenguaje, aceptado como norma ISO desde el año 2000.

HTML fue diseñado originalmente para intercambiar información en entornos académicos, sus etiquetas están eminentemente pensadas para la organización lógica del contenido (titulo, párrafo, etc.) y no tanto para su presentación. Por eso, el W3C tuvo que idear las denominadas hojas de estilo (*Style Sheets*).

En el 2000, aparece el **XML** (*Extensible Markup Language*) como complemento al HTML. XML y HTML son lenguajes muy diferentes. Su función principal es ayudarnos a organizar contenidos y eso hace que los documentos XML sean portables hacia diferentes tipos de aplicaciones. Aunque ambos se basan en el SGML, cada uno ha sido diseñado para cumplir distintas funciones: el XML sirve para estructurar información de cualquier tipo mientras que el HTML sirve para estructurar el contenido de texto de un documento web y su metainformación y presentarla a través de un navegador.

Posteriormente surge **XHTML** (*eXtensible HyperText Markup Language*), diseñado para extender HTML y permitir la compatibilidad con nuevos formatos de datos como videos, imágenes o lenguajes de scripting. A finales de 2009 se interrumpe el trabajo sobre la versión 2.0 porque el *Grupo de Trabajo de Tecnología de Aplicación de Hipertexto Web* (WHATWG) estaba trabajando en **HTML5** al mismo tiempo, y este último fue el estándar que al final prevaleció. Sin embargo, HTML 5 ya no es una aplicación de SGML, sino un lenguaje generalizado compatible con las versiones anteriores.

2.3. Cómo idear un lenguaje de marcas

Todo lenguaje de marcas debe tener dos componentes principales:

- Un **vocabulario** o conjunto de elementos a utilizar
- Unas **reglas gramaticales** que indiquen cómo colocar estos elementos

Supongamos que queremos almacenar la información de los libros que tenemos en casa. Para cada libro queremos almacenar su título, su autor y su número de páginas. Podríamos definir un documento con las etiquetas `libro`, `titulo`, `autor` y `paginas` como vocabulario, además de la etiqueta `biblioteca` para englobarlo todo. Adicionalmente, de los autores nos puede interesar saber su año de nacimiento, si lo conocemos. Gramaticalmente, la biblioteca se compone de libros, y cada libro de sus atributos básicos (título, autor y número de páginas), con lo que podríamos conformar un documento así:

```
<biblioteca>
    <libro>
        <titulo>El juego de Ender</titulo>
        <autor>Orson Scott Card</autor>
        <paginas>325</paginas>
    </libro>
    <libro>
        <titulo>La tabla de Flandes</titulo>
        <autor nacimiento="1951">Arturo Pérez Reverte</autor>
        <paginas>384</paginas>
    </libro>
</biblioteca>
```

En este primer ejemplo, podemos observar varias cosas:

- Los elementos que componen el vocabulario de nuestro documento (etiquetas `libro`, `titulo`, etc), llamados en este caso **etiquetas**.
- Que cada etiqueta puede tener un elemento de apertura y uno de cierre, que delimita el contenido afectado por esa etiqueta
- Que algunas etiquetas pueden contener **atributos**, como es el caso del año de nacimiento del autor, para especificar información adicional.
- Las reglas gramaticales indican qué etiquetas se pueden colocar en cada ámbito. Por ejemplo, no deberíamos poder poner un título si no es dentro de un libro.

3. Características y tipos de lenguajes de marcas

Veamos ahora qué características generales debe cumplir cualquier lenguaje de marcas que definamos o utilicemos.

- Deben definir la información en archivos de **texto plano**, de forma que sean fácilmente exportables o utilizables entre distintas plataformas o sistemas.
- Deben mostrar la información de la forma más **compacta** posible, intercalando las etiquetas o palabras del vocabulario con la información que se quiere almacenar.
- Los lenguajes de marcas a menudo deben coexistir con otros lenguajes en un documento. Es lo que ocurre, por ejemplo, con el lenguaje HTML, que a menudo convive con fragmentos de código en lenguaje JavaScript, o PHP, entre otros.
- Los lenguajes de marcas **no son lenguajes de programación**, ya que carecen de las estructuras básicas de estos lenguajes, como el uso de variables, bucles, funciones, etc. Son lenguajes para etiquetar y estructurar la información de documentos, sin más.

3.1. Tipos de lenguajes de marcas

Existen diferentes tipos de lenguajes de marcas. Veamos aquí las principales categorías:

- **Lenguajes de presentación**: permiten definir el formato con que se ve la información (tipo de letra, tamaño, color, etc.), sin entrar en detalles sobre su estructura. Un ejemplo de lenguaje en esta categoría sería **RTF**.
- **Lenguajes descriptivos, estructurales o semánticos**: permiten definir las partes en que se estructura un documento, sin especificar cómo deben mostrarse, ni en qué orden. El ejemplo más representativo de esta categoría es **XML**, incluyendo todos los tipos que derivan de él. También otros lenguajes, como **YAML** o **JSON**, empleados para definir ficheros de configuración en algunas aplicaciones, corresponden a esta categoría.
- **Lenguajes híbridos**: lenguajes que encajan en las dos categorías anteriores: permiten definir la estructura de un documento y también, de forma total o parcial, su formato de presentación. El ejemplo más representativo de esta categoría es el lenguaje **HTML**, incluyendo algunas de sus variantes como **xHTML**.

Según su **funcionalidad** también podemos clasificar los lenguajes de marcas en distintas categorías:

- Lenguajes para generar documentación electrónica: RTF, TeX, DocBook...
- Lenguajes para tecnologías de Internet: HTML o xHTML para definir páginas web con su contenido, RSS para sindicar noticias, SOAP para definir la información proporcionada por servicios web...
- Lenguajes de propósito específico para ciertos tipos de aplicaciones: SVG para definir imágenes vectoriales, FXML para definir interfaces gráficas en Java utilizando JavaFX, XSLT para transformar documentos XML, etc.

El origen de todos estos lenguajes es, como hemos visto, el lenguaje SGML. A partir de él se crearon lenguajes como HTML o XML, entre otros, y a partir de éstos, se derivaron otros.

4. Herramientas

A la hora de trabajar con un determinado lenguaje de marcas (o con un conjunto de ellos) podemos valernos de ciertas herramientas software que nos faciliten el proceso. Veremos a continuación algunas de ellas, y explicaremos cuál es la que vamos a utilizar a lo largo de este curso.

4.1. Notepad++

Notepad++ es una herramienta sencilla y gratuita, que permite editar archivos de texto plano en diferentes lenguajes (tanto de marcas como de programación). Podemos descargarlo desde su web oficial, pero su uso está restringido a sistemas Windows.

4.2. Visual Studio Code

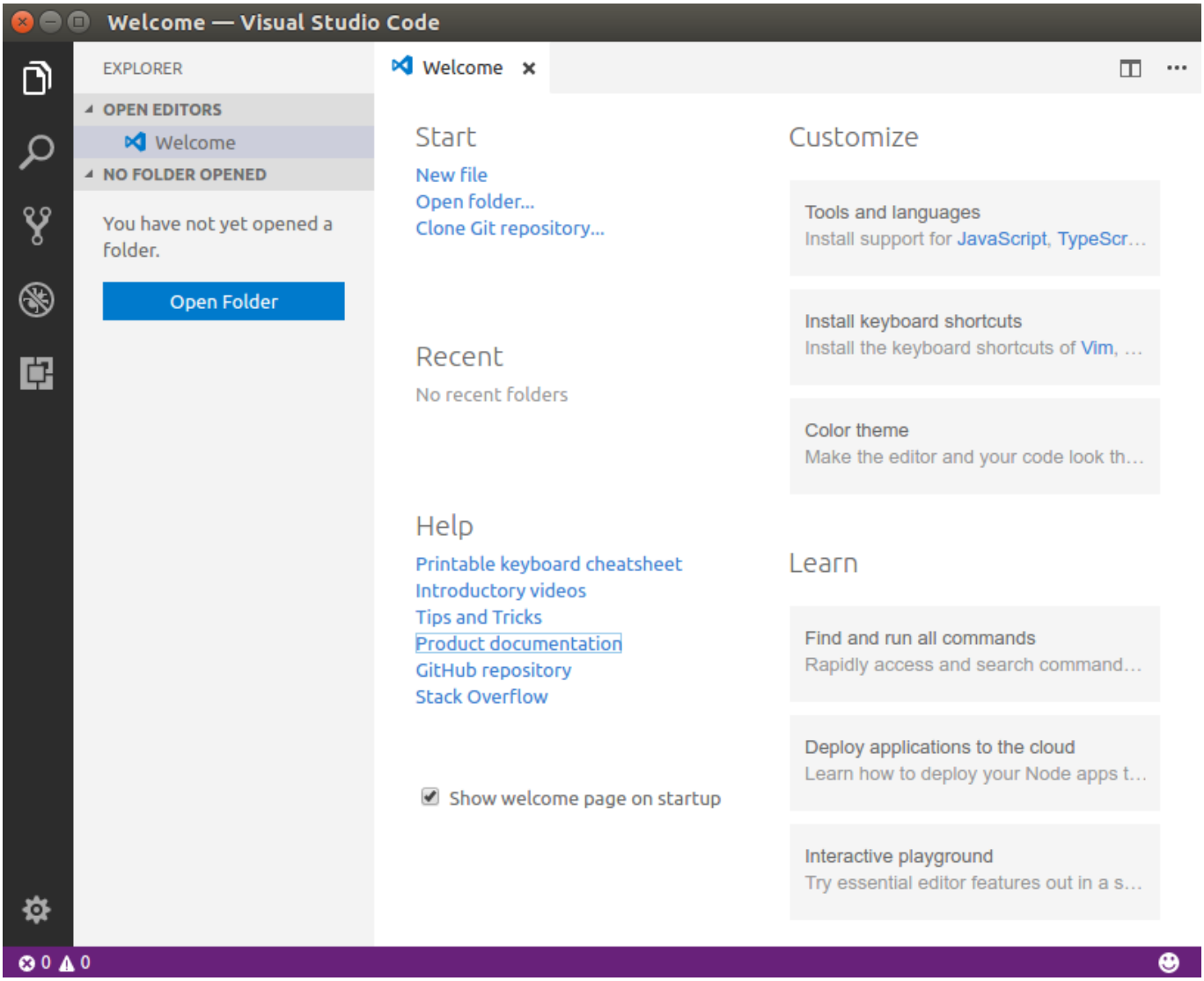
Visual Studio Code es un editor ligero y potente, disponible para Windows, MacOSX y Linux. Tiene soporte nativo para lenguajes de programación derivados de JavaScript (JavaScript, TypeScript, Node.js...), pero también dispone de una gran variedad de *plugins* o extensiones para poder trabajar con otros muchos lenguajes. Será el editor que utilizaremos en este curso.

Podemos descargarlo de su web oficial. En el caso de Windows o Mac OSX descargamos un asistente de instalación que nos guía paso a paso en el proceso. En el caso de Linux, por ejemplo para usuarios Debian, Ubuntu o derivados, descargaremos un archivo *.deb* que deberemos instalar manualmente. Para ello, iremos a la carpeta donde lo hayamos descargado y escribiremos este comando:

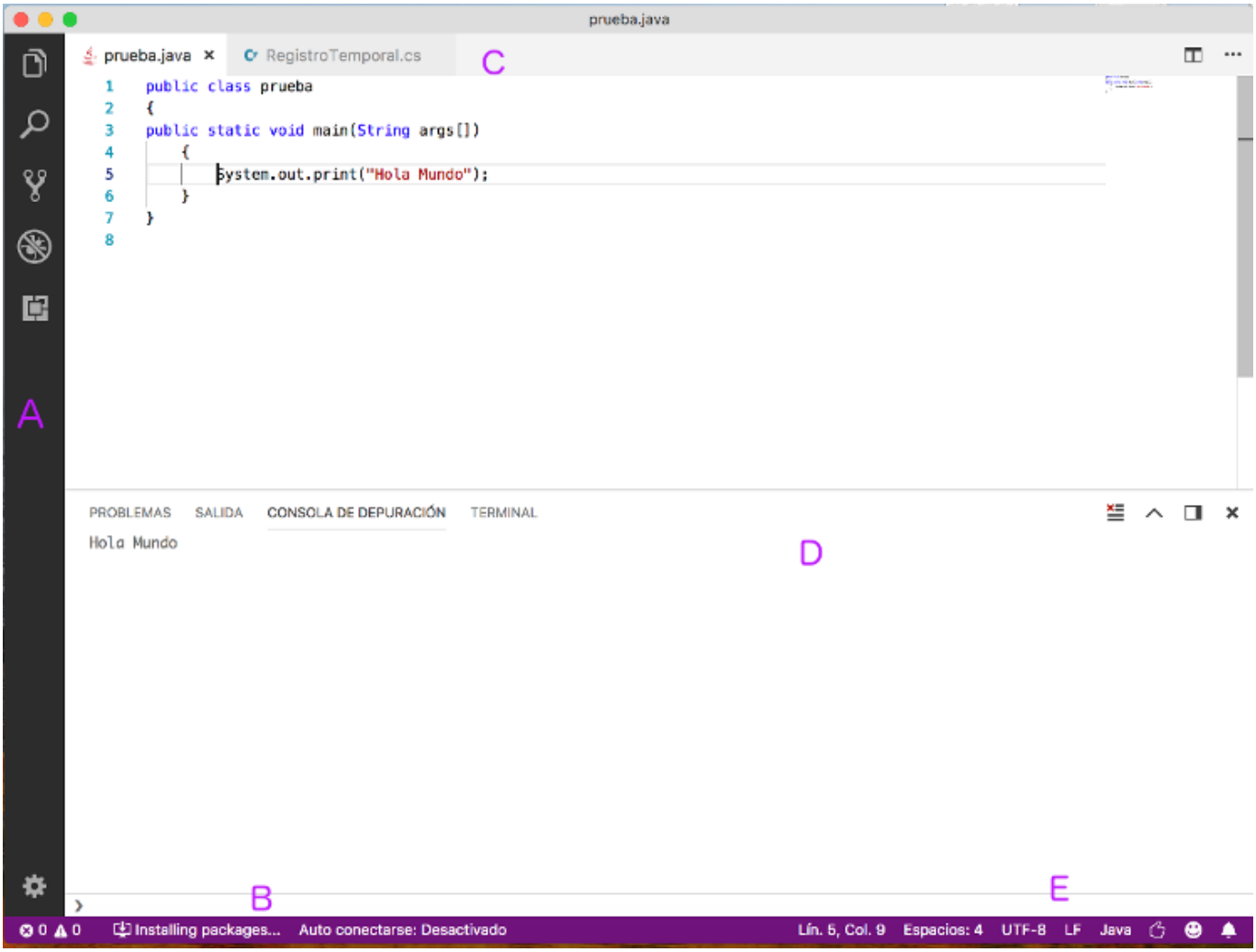
```
sudo dpkg -i <nombre_archivo.deb>
```

4.2.1. Entorno de trabajo

Una vez instalado, podemos ver una pantalla de bienvenida con ciertas opciones básicas:



Si creamos o abrimos un archivo, podremos ver el entorno de trabajo habitual, dividido en varias secciones:



- **Editor** (C): Sección de edición principal, donde tendremos abiertos nuestros archivos para editarlos. Podemos tener abiertos tantos como queramos, cada uno en una pestaña diferente.
- **Barra de margen** (B): Contiene cierta información relevante sobre el explorador de archivos, errores en el texto, advertencias, etc.
- **Barra de estado** (E): Muestra información sobre el proyecto actualmente abierto, y los archivos que estamos editando (línea actual, codificación del archivo, etc.).
- **Barra de actividades** (A): Dispone de ciertos iconos o herramientas principales. Al pulsarlos, se abrirá/ocultará, alternativamente, la opción correspondiente de este panel.

    - **Explorador**: para gestionar los archivos y carpetas de nuestro proyecto (o la carpeta que tengamos abierta actualmente)
    - **Búsqueda**: para buscar palabras o expresiones en los archivos abiertos.
    - **Control de versiones**: para poder comunicar y subir los archivos a un repositorio remoto tipo GitHub o similar.
    - **Depurador**: para depurar los programas y detectar posibles fallos
    - **Gestor de extensiones**: desde aquí podremos instalar funcionalidades adicionales sobre VS Code, y adaptarlo a los lenguajes con los que vayamos a trabajar.

- **Paneles** (D): Bajo el editor tenemos diferentes paneles, identificados por pestañas. Por ejemplo, tenemos un panel para ver los errores o advertencias sobre el código o texto que estamos editando, o un terminal

para poder ejecutar comandos desde la carpeta donde estamos trabajando.

Cada vez que abramos VS Code, veremos los archivos y el estado en que lo dejamos la última vez que lo utilizamos.

4.2.2. Cambiar el tema

Si queremos cambiar el tema por defecto con que se instala Visual Studio Code, debemos ir al menú *File/Preferences* (o *Code/Preferences* si estamos en Mac OSX) y elegir la opción *Color Theme*. Ahí podremos elegir entre diferentes estilos, aunque los más habituales o populares son los temas *Dark* y *Light*.

4.2.3. Instalando extensiones

Una de las características más destacables e importantes de Visual Studio Code es la posibilidad de mejorar su funcionamiento y usabilidad instalando extensiones adicionales. Muchas de estas extensiones aluden a un lenguaje en concreto, facilitando su uso a través de auto completados, comprobaciones de sintaxis, etc.

Para instalar extensiones, debemos ir al icono de extensiones en el panel izquierdo, y buscar la que queramos.

Por ejemplo, podemos instalar una extensión llamada *open in browser*, desarrollada por *TechER*, que permite abrir documentos HTML directamente en el navegador, para previsualizarlos. Simplemente elegimos la extensión, y pulsamos el botón de *Install*.

Para poderla utilizar, cuando estemos editando un archivo HTML podemos hacer clic derecho sobre el código y elegir la opción *Open in Default Browser*.

> **Ejercicio 1:**
>
> Crea un archivo llamado `prueba.html`. Ábrelo con VS Code y edítalo con este contenido:

```
<!DOCTYPE html>
<html>
    <body>
        <h1>Bienvenido/a a la página de prueba</h1>
    </body>
</html>
```

> Cuando lo tengas listo, haz clic derecho en el código fuente y muestra la previsualización en tu navegador predeterminado, usando la extensión anterior.

> **Ejercicio 2:**

> Crea un archivo llamado `biblioteca.xml`. Ábrelo con VS Code y edítalo con este contenido:

```
<biblioteca>
    <libro>
        <titulo>El juego de Ender</titulo>
        <autor>Orson Scott Card</autor>
        <paginas>325</paginas>
    </libro>
    <libro>
        <titulo>La tabla de Flandes</titulo>
        <autor nacimiento="1951">Arturo Pérez Reverte</autor>
        <paginas>384</paginas>
    </libro>
</biblioteca>
```

> Cuando lo tengas listo, ábrelo en el navegador y observa cómo se muestra la información en este caso.

4.3. Otras alternativas

Además de Visual Studio Code, existen otras alternativas más o menos similares que podemos emplear en distintas plataformas. Muchas de estas herramientas siguen una filosofía similar a VS Code, en cuanto a que podemos gestionar proyectos basados en carpetas (abrir una carpeta y gestionar todos los archivos que contiene desde el editor), e instalar plugins adicionales para adaptar la herramienta al lenguaje o lenguajes con que trabajemos.

Algunos de los ejemplos más representativos de estas herramientas son los editores **Sublime Text** (web oficial) o **Atom** (web oficial).