Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Desarrollo de una herramienta de apoyo a la planificación horaria de la ENM

Grado en Ingeniería Mecánica

ALUMNO: Julio Albaladejo Carreño

DIRECTOR: Miguel Rodelgo Lacruz

CURSO ACADÉMICO: 2018-2019

Universidade Vigo

Centro Universitario de la Defensa
en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Desarrollo de una herramienta de apoyo a la planificación horaria de la ENM

Grado en Ingeniería Mecánica

Intensificación en Tecnología Naval

Cuerpo General / Infantería de Marina

Universidade Vigo

RESUMEN

Teniendo en cuenta la complejidad de la planificación horaria de la ENM, se ve de gran ayuda contar con una herramienta que revise posibles fallos como la sobreasignación de profesorado y permita realizar consultas como el horario específico de un profesor de manera rápida e intuitiva. Haciendo uso del lenguaje de programación Python y bases de datos relacionales en formato SQL, se ha realizado un programa que consigue ambos objetivos.

La primera parte de dicha herramienta, haciendo uso del módulo externo Openpyxl para Python, extrae los datos necesarios de los archivos Excel que contienen el horario general de la ENM y las planificaciones específicas de cada asignatura. Una vez obtenidos los datos, se almacenan en una base de datos en formato SQL. Por último, la interfaz del programa permite al usuario consultar y gestionar la base de datos creada sin tener conocimiento de los comandos SQL o, en caso de conocerlos, tiene también la opción de utilizarlos.

PALABRAS CLAVE

Planificación, Horarios, Python, SQL, Consulta

AGRADECIMIENTOS

Quiero agradecer a todas las personas que han formado parte de mi paso por la Escuela Naval Militar por su paciencia y dedicación a la enseñanza. También quiero agradecer a todos los compañeros que han convivido conmigo durante los buenos y los malos momentos. Por último, quiero agradecerle a mi abuelo el haber estado siempre ahí para cualquier cosa que he necesitado.

CONTENIDO

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Antecedentes

En la Escuela Naval Militar se imparte la formación necesaria para los oficiales que la Armada necesita. Esta formación incluye un Título de Grado Universitario impartido por el Centro Universitario de la Defensa, formación militar general, formación específica de cada cuerpo y actividades de instrucción y adiestramiento.

Actualmente conviven más de diez planes de estudio diferentes para los cuales se necesita un gran número de aulas y profesores. A su vez, los planes de estudio con más número de alumnos se dividen en grupos de clase. Para organizar todos estos planes de estudio y grupos de clase existe un horario semanal, denominado coloquialmente "sábana", en el que aparecen todas las clases y actividades de cada grupo. Hay dos tipos de semana que se repiten de manera alternativa: semanas tipo A y semanas tipo B. Para semanas específicas, como semanas con festivos, se crean sábanas exclusivas.

Alumnos y profesores deben consultar regularmente dichas sábanas para conocer las horas y las aulas en las que tendrán clase. Además, también deben consultar otro documento que se publica diariamente (Orden Diaria de la ENM) para comprobar si hay algún cambio de última hora en las clases o actividades del día en cuestión.

1.2 Motivación

Como en cualquier universidad, colegio o instituto, la realización de un horario que contenga todas las asignaturas, profesores y cursos o grupos de clase, es muy compleja. En la Escuela Naval Militar, el problema se acentúa al compaginar las asignaturas del Grado en Ingeniería Mecánica con las asignaturas militares y actividades de instrucción y adiestramiento además de los continuos imprevistos que surgen cada semana (actos, conferencias, etc.). Esto obliga a los responsables de la creación del horario a realizar numerosos cambios incluso llegando al extremo de tener que crear una sábana específica para una semana concreta. La enorme complejidad del proceso es la que hace que a menudo se generen errores (ver Figura 1-1).

CURSO 2018-2019 SEGUNDO CUATRIMESTRE		CUERPO GENERAL						INFANTERÍA MARINA	
		GM2 (CGA) PROM. 420						GM2 (CIM) PROM. 150	
		G1	G2	G3	G4	G5	G6	IM1	IM2
MARTES	0845-0935	ACTIVIDADES DE RÉGIMEN INTERIOR							
	0940-1030	FORMACIÓN MILITAR III G1G2G3 TCOL_LÓPEZ ALVES 45			SEGURIDAD INTERIOR I G4G5G6 CC_VARELA SÁNCHEZ 56			TOPO_CONST IM1 54	SIS_RADIO IM2 53
	1050-1140	SEGURIDAD INTERIOR I G1G2G3 CC_VARELA SÁNCHEZ 45			SISTEMAS DE ARMAS Y TN I G4G5G6 TN_MEJÍAS MENDOZA 56			SIS_RADIO IM1 53	TOPO_CONST IM2 54
	1145-1235	MÁQ_MOT G1 45	SIS_RADIO G2 56	DISEÑO_MÁQ. G3G4 3		ING_FAB_CAL_DIM G5G6 3		SISTEMAS ARMAS IM CAP_VICTORIA FERNÁNDEZ 54	
	1240-1330	SIS_RADIO G1 56	MÁQ_MOT G2 45	DISEÑO_MÁQ. G3G4 3		ING_FAB_CAL_DIM G5G6 3		SISTEMAS ARMAS IM CAP_VICTORIA FERNÁNDEZ 54	
	1335-1425	ESTUDIO	ESTUDIO	ESTUDIO	ESTUDIO	SIS_RADIO G5 56	MÁQ_MOT G6 45	TÁCTICA ANFIBIA III CTE_DÍEZ GONZÁLEZ 54	
	1550-1640	PRÁCTICAS EN LA MAR G1G2		SIMTAC G3	INS MARINERA G4	DISEÑO_MÁQ. G5G6 3		ING_FAB_CAL_DIM IM 18	
	1645-1735					DISEÑO_MÁQ. G5G6 3		ING_FAB_CAL_DIM IM 18	
	1740-1830			ESTUDIO		MÁQ_MOT G5 45	SIS_RADIO G6 56	ESTUDIO	ESTUDIO

Figura 1-1 Ejemplo de error de tipo solape: a las 11:45 hay dos clases diferentes programadas para el aula 3

Aun sin contemplar la posibilidad de errores, filtrar toda la información de la extensa sábana y contrastarla con los demás documentos referentes al horario es una tarea ardua. Esto afecta tanto a los que leen la sábana como a los encargados de modificarla.

En proyectos anteriores, se consideró atacar el problema en la raíz, buscando una herramienta fiable y flexible que permitiese la creación y edición de la sábana de manera automatizada. Las herramientas de este tipo son muy complejas y requieren una rigurosa implementación de restricciones por parte del usuario. Al no alcanzar resultados satisfactorios, se decide continuar haciendo la sábana de manera manual y tratar el problema a partir del horario ya creado, es decir, encontrar la manera de postprocesar la sábana para poder detectar posibles solapes y realizar consultas.

1.3 Objetivos

El objetivo general de este proyecto es crear una herramienta capaz de interpretar la sábana de la Escuela Naval Militar, almacenar la información de manera organizada y poder realizar consultas que estén libres de errores. El programa a realizar debe ser intuitivo y accesible para todos los profesores y alumnos que lo deseen.

Para alcanzar este objetivo general se proponen los siguientes objetivos específicos:

- Breve estado del arte para poder elegir acertadamente las tecnologías más apropiadas para el desarrollo de la herramienta.
- Diseño del parser, modelo de datos e interfaz de la herramienta.
- Implementación de la herramienta.
- Pruebas y validación de la herramienta.

2 ESTADO DEL ARTE

2.1 Formato del horario actual

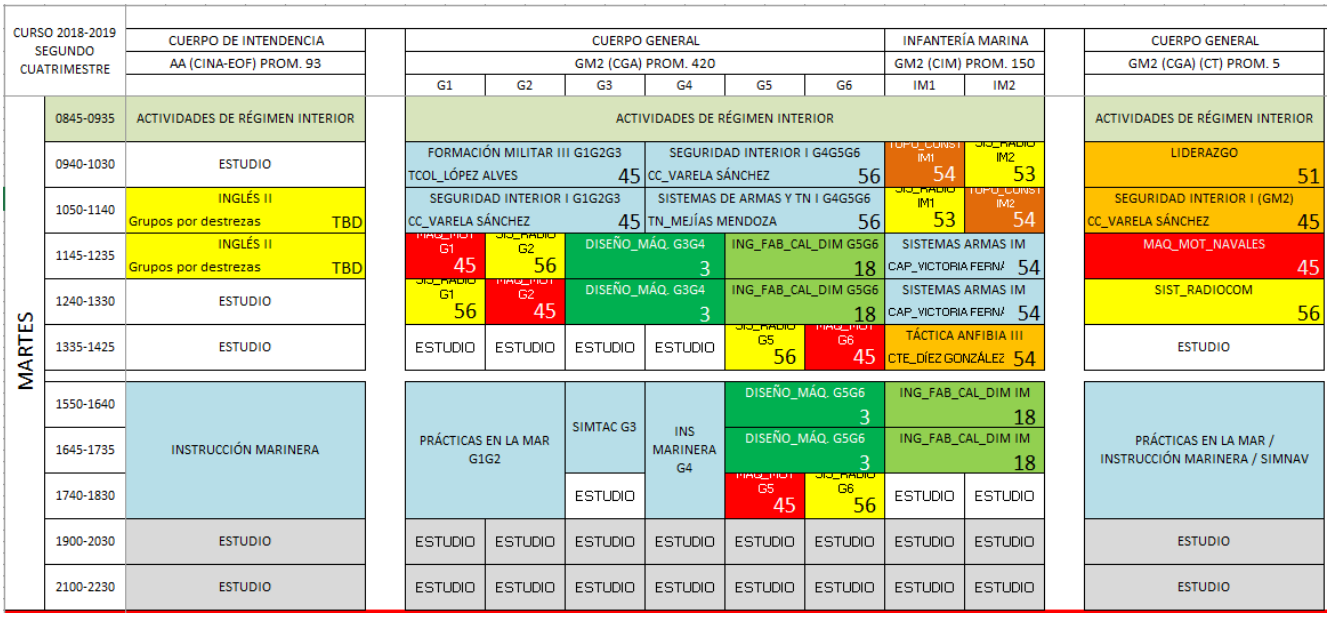
En este apartado se expondrá el método actual que se sigue para organizar los horarios de alumnos y profesores. Se explicará la estructura de los documentos que se utilizan para este fin.

2.1.1 Sábana general

El documento que contiene los horarios de cada grupo de clase se publica al comienzo de cada cuatrimestre. Se genera en formato de tabla a través del programa Microsoft Excel. Consta de dos hojas, una con la tabla correspondiente a la semana tipo A y otra con la tabla de la semana tipo B.

La estructura de las tablas se compone de los cursos y grupos en la parte superior (eje de abscisas) y los días de la semana y horas en el lateral izquierdo (eje de ordenadas). Las columnas están ordenadas por antigüedad de izquierda a derecha, es decir, los alumnos con empleos superiores tienen su horario a la izquierda de los alumnos más modernos. Las filas están ordenadas siguiendo un orden cronológico de arriba abajo.

En cada celda de la tabla encontramos el nombre de la asignatura en la mitad superior, el aula en la esquina inferior derecha y, en algunos, casos el nombre del profesor en el espacio restante. El color de las celdas va asociado con cada asignatura, aunque no todas las asignaturas tienen colores exclusivos, colores como el amarillo, el azul o el naranja se emplean para numerosas asignaturas. Todo lo mencionado se puede apreciar en la Figura 2-1.



CURSO 2018-2019 SEGUNDO CUATRIMESTRE		CUERPO DE INTENDENCIA AA (CINA-EOF) PROM. 93	CUERPO GENERAL GM2 (CGA) PROM. 420 G1	CUERPO GENERAL GM2 (CGA) PROM. 420 G2	CUERPO GENERAL GM2 (CGA) PROM. 420 G3	CUERPO GENERAL GM2 (CGA) PROM. 420 G4	CUERPO GENERAL GM2 (CGA) PROM. 420 G5	CUERPO GENERAL GM2 (CGA) PROM. 420 G6	INFANTERÍA MARINA GM2 (CIM) PROM. 150 IM1	INFANTERÍA MARINA GM2 (CIM) PROM. 150 IM2	CUERPO GENERAL GM2 (CGA) (CT) PROM. 5
MARTES	0845-0935	ACTIVIDADES DE RÉGIMEN INTERIOR	ACTIVIDADES DE RÉGIMEN INTERIOR								ACTIVIDADES DE RÉGIMEN INTERIOR
	0940-1030	ESTUDIO	FORMACIÓN MILITAR III G1G2G3 TCOL_LÓPEZ ALVES 45			SEGURIDAD INTERIOR I G4G5G6 CC_VARELA SÁNCHEZ 56			TOPO_CONST IM1 54	SIS_RADIO IM2 53	LIDERAZGO 51
	1050-1140	INGLÉS II Grupos por destrezas TBD	SEGURIDAD INTERIOR I G1G2G3 CC_VARELA SÁNCHEZ 45			SISTEMAS DE ARMAS Y TN I G4G5G6 TN_MEJÍAS MENDOZA 56			SIS_RADIO IM1 53	TOPO_CONST IM2 54	SEGURIDAD INTERIOR I (GM2) CC_VARELA SÁNCHEZ 45
	1145-1235	INGLÉS II Grupos por destrezas TBD	MAQ_MOT G1 45	SIS_RADIO G2 56	DISEÑO_MÁQ. G3G4 3		ING_FAB_CAL_DIM G5G6 18		SISTEMAS ARMAS IM CAP_VICTORIA FERNA 54		MAQ_MOT_NAVALES 45
	1240-1330	ESTUDIO	SIS_RADIO G1 56	MAQ_MOT G2 45	DISEÑO_MÁQ. G3G4 3		ING_FAB_CAL_DIM G5G6 18		SISTEMAS ARMAS IM CAP_VICTORIA FERNA 54		SIST_RADIOCOM 56
	1335-1425	ESTUDIO	ESTUDIO	ESTUDIO	ESTUDIO	ESTUDIO	SIS_RADIO G5 56	MAQ_MOT G6 45	TÁCTICA ANFIBIA III CTE_DÍEZ GONZÁLEZ 54		ESTUDIO
	1550-1640	INSTRUCCIÓN MARINERA	PRÁCTICAS EN LA MAR G1G2		SIMTAC G3	INS MARINERA G4	DISEÑO_MÁQ. G5G6 3		ING_FAB_CAL_DIM IM 18		PRÁCTICAS EN LA MAR / INSTRUCCIÓN MARINERA / SIMNAV
	1645-1735						DISEÑO_MÁQ. G5G6 3		ING_FAB_CAL_DIM IM 18		
	1740-1830				ESTUDIO		MAQ_MOT G5 45	SIS_RADIO G6 56	ESTUDIO	ESTUDIO	
	1900-2030	ESTUDIO	ESTUDIO	ESTUDIO	ESTUDIO	ESTUDIO	ESTUDIO	ESTUDIO	ESTUDIO	ESTUDIO	ESTUDIO
	2100-2230	ESTUDIO	ESTUDIO	ESTUDIO	ESTUDIO	ESTUDIO	ESTUDIO	ESTUDIO	ESTUDIO	ESTUDIO	ESTUDIO

Figura 2-1 Fragmento de la sábana general de la ENM

2.1.2 Planificación específica de cada asignatura

Para organizarse internamente en cada asignatura, los profesores deben crear otro documento en el que se distribuyen las horas de clase de cada semana del curso. Dicho documento se realiza también con el programa Microsoft Excel. Cabe reseñar que las asignaturas militares no se organizan de esta manera y por tanto no hacen uso de este documento.

El documento debe contener el número de clases de teoría, seminarios y prácticas que imparte cada profesor de la asignatura (Figura 2-2). La estructura es la misma para todas las asignaturas.

PROFESOR/ES	SEMANA 13 (1-5 abr) T	P	S	TOTAL	SEMANA 14 (8-12 abr) T	P	S	TOTAL
Santiago Urréjola Madriñán	4			4	4			4
Rosa Devesa Rey			8	8				0
Victor Alfonsín Pérez				0		8		8
0				0				0
0				0				0
0				0				0
TOTAL	4	0	8	**12**	4	8	0	**12**

Figura 2-2 Fragmento del documento de planificación de Química

2.2 Lenguajes de programación

A la hora de elegir un lenguaje de programación se encuentra una oferta muy variada. Recurrir a los lenguajes más usados suele ser una buena idea dado que se encontrará más documentación y soporte. A continuación, se expondrán los cuatro lenguajes más usados actualmente.

2.2.1 Java

Hoy en día es lenguaje más usado mundialmente como se puede apreciar en la Figura 2-3. Fue inventado por Sun Microsystems en 1996 con el objetivo de crear un lenguaje multiplataforma. Para conseguir dicho objetivo se optó por depender de un intérprete, es decir, usar un programa de intermediario a la hora de ejecutar el código. Su sintaxis es similar a la de C o C++. Es un lenguaje

orientado a objetos y con tipado estático, esto obliga a declarar el tipo de variable que se desea usar. Es robusto y seguro. Su característica más valorada es el recolector de basura que permite reutilizar los espacios de memoria RAM que van quedando en desuso a medida que se ejecuta un programa. [1]

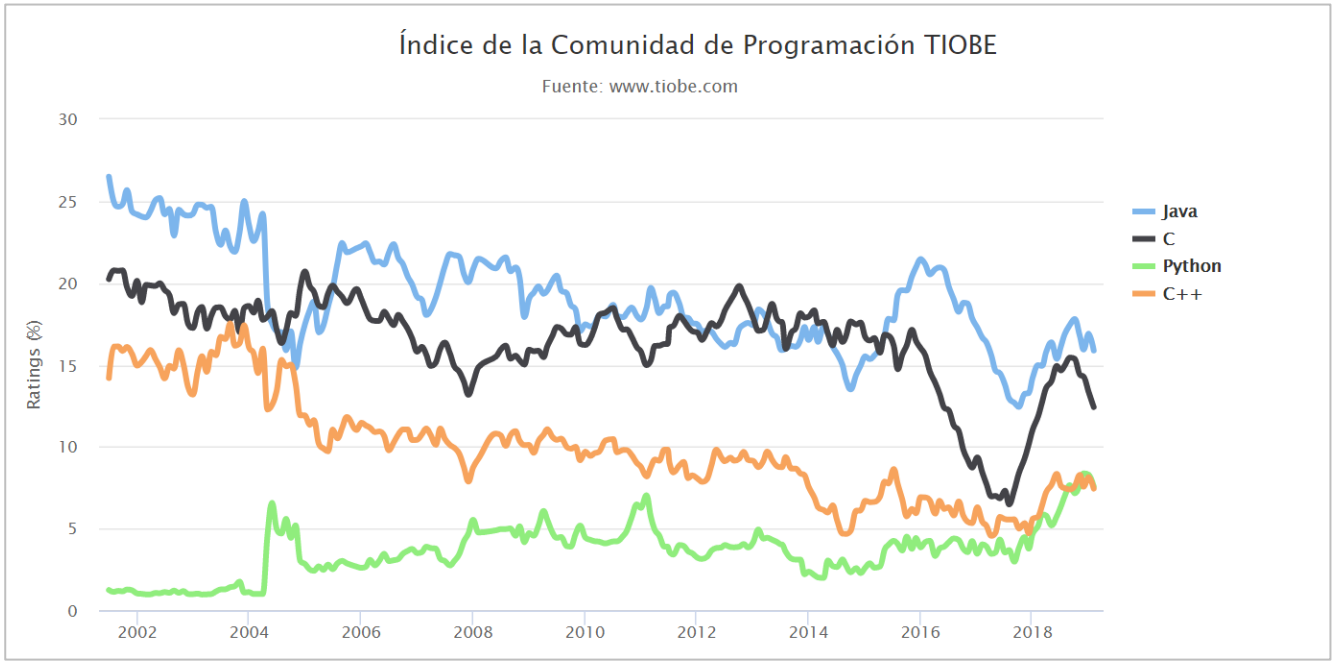


Figura 2-3 Comparativa de índices TIOBE en el tiempo [2]

2.2.2 C

Es un lenguaje de nivel intermedio creado en 1972 por Denis Ritchie. Este nivel intermedio le permite tener el rendimiento de los lenguajes de bajo nivel a la vez que la sencillez de los lenguajes de alto nivel. Entre sus múltiples aplicaciones se encuentra la creación de sistemas operativos ya que con este lenguaje se tiene un control absoluto de lo que ocurre en las entrañas de un ordenador. Necesita de un compilador que traduzca el código a lenguaje de máquina previamente a la ejecución. No permite programación orientada a objetos y usa punteros para acceder a la memoria. Es un lenguaje complejo para aprender y poder sacarle un rendimiento apropiado. [3]

2.2.3 Python

Se trata de un lenguaje de programación muy potente, creado en 1991 por Guido van Rosum, cuya popularidad no para de aumentar como se puede apreciar en Figura 2-3. Es un lenguaje interpretado, es decir, no necesita compilarse para ser ejecutado ya que lo hace a través de un programa llamado intérprete. El tipado es dinámico por lo que no se tiene que declarar el tipo de variable que se va a emplear. Tiene la ventaja de que el intérprete existe para múltiples plataformas de manera que no hay que modificar el código si se cambia de sistema operativo además de que está orientado a objetos. Cabe destacar el amplio abanico de funciones predeterminadas que ofrece a la hora de operar con listas y cadenas de texto. [4]

2.2.4 C++

Diseñado por Bjarne Stroustrup en 1979 como una versión de C con el añadido de ser orientado a objetos. Necesita ser compilado a lenguaje de máquina para poder ser ejecutado, lo que le hace depender de un programa compilador. Es un lenguaje fuertemente tipado, es decir, hay que ser preciso a la hora de declarar una variable. Usa punteros y alcanza una gran eficiencia al ser un lenguaje de más bajo nivel. [5]

2.3 Bibliotecas para manejo de archivos de Microsoft Excel

Las bibliotecas aportan funciones y objetos con sus correspondientes métodos y atributos que aumentan las capacidades del lenguaje de programación en cuestión. En este caso se expondrán bibliotecas de Python relacionadas con el manejo de archivos de Microsoft Excel.

2.3.1 Pandas

Esta biblioteca está pensada para trabajar con tablas de datos en formato .csv. También se pueden leer y escribir tablas en Microsoft Excel, sin embargo, el abanico de operaciones disponibles es limitado. Está enfocado a operaciones con tablas sencillas que solo contengan números. Para aprovecharla al máximo se debe complementar con otra biblioteca llamada *NumPy*. [6]

2.3.2 Openpyxl

El objetivo principal de la creación de esta biblioteca fue operar con hojas de cálculo en formato .xlsx (Microsoft Excel) además de otros formatos Office Open XML. Con ella se pueden leer y escribir archivos Excel casi sin restricciones en comparación con las funcionalidades que ofrece la interfaz de Microsoft. Entre las prestaciones que ofrece esta biblioteca, se encuentra la posibilidad de operar con formatos condicionales o proteger hojas de cálculo con contraseña. [7]

2.4 Sistemas gestores de bases de datos relacionales

El principal objetivo de las bases de datos es gestionar grandes cantidades de información de manera ordenada y evitando redundancias. Éstas pueden almacenarse localmente en un equipo o en un servidor accesible por diferentes usuarios. En estos sistemas la consistencia de los datos es primordial, por tanto, incluso con varios usuarios realizando consultas o modificaciones, el sistema mantiene dicha condición. Existen diferentes modelos para estructurar la información: relacional, jerárquico, en red, etc.

La mayoría de los sistemas gestores de bases de datos optan por el modelo relacional. Este tipo de modelo organiza la información en tablas. Cada tabla contiene filas o tuplas que representan cada registro y columnas que definen los atributos de cada registro. Para evitar la duplicidad de registros debe existir una columna denominada clave, los atributos correspondientes a dicha columna deben ser únicos de manera que identifiquen al registro de manera unívoca. [8]

Para crear, consultar y modificar las bases de datos se utiliza el lenguaje SQL (*Structured Query Language* o lenguaje de consulta estructurada en español). Algunos de los comandos más básicos de este lenguaje son los siguientes:

- *CREATE* para crear una nueva base de datos o tabla.
- *INSERT INTO* para crear un nuevo registro en una tabla.
- *UPDATE* para modificar un registro.
- *SELECT* para realizar una consulta.

Los tres grandes desarrolladores de este tipo de sistemas Oracle, MySQL y Microsoft SQL Server son un ejemplo de modelo relacional. Cada motor de bases de datos está enfocado a un uso concreto y se debe tener en cuenta a la hora de escoger uno.

2.4.1 MySQL

Creado en 1995 usando una mezcla de código en C y C++, es uno de los sistemas de gestión de bases de datos más populares del mundo (Figura 2-4). Su código es abierto y gestiona bases de datos relacionales a través de comandos SQL. Está enfocado a la gestión de una base de datos situada en un servidor. Actualmente es el software que se emplea en la ENM para practicar el uso de bases de datos en la asignatura de *Informática para la Ingeniería*. Ofrece una interfaz para crear y editar bases de datos de manera gráfica y otra a través de una línea de comandos. [9]

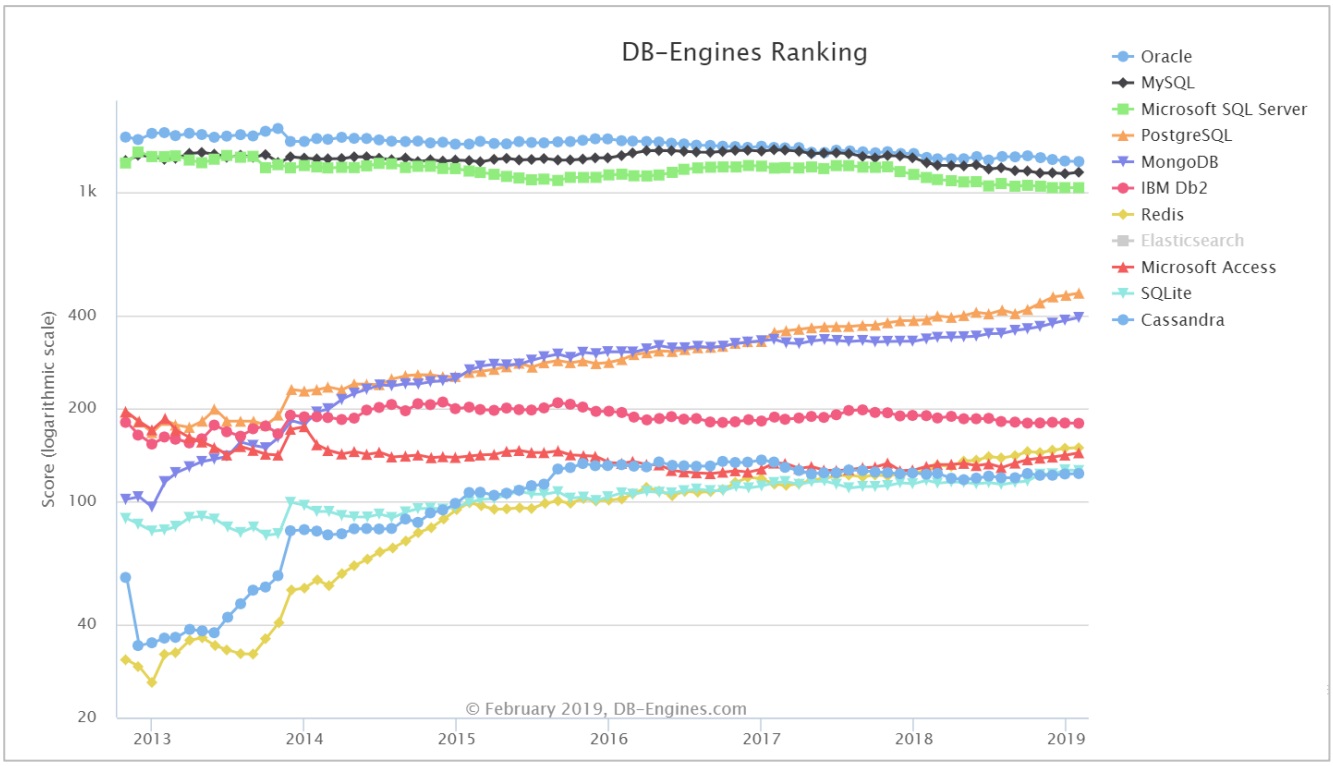


Figura 2-4 Ranking de motores de bases de datos [10]

2.4.2 SQLite

Lanzado en el año 2000 por D. Richard Hipp, es un sistema minimalista de gestión de bases de datos relacionales. Se trata de software libre por lo que no requiere una licencia para su utilización. A pesar de tener menos funcionalidades que sus tres grandes competidores, sigue siendo muy utilizado para pequeñas aplicaciones que requieren alguna funcionalidad de una base de datos. Los datos se almacenan en un fichero local en formato .db lo que facilita transferir la base de datos de una terminal a otra. Además, entre las bibliotecas que ofrece Python por defecto, se encuentra una específica para el manejo de este tipo de bases de datos. [11]

3 DESARROLLO DEL TFG

3.1 Elección de herramientas de trabajo

3.1.1 Lenguaje de programación

De entre los cuatro lenguajes expuestos en el estado del arte de la presente memoria, se optó por el uso de Python. Al ser de alto nivel, resulta más sencillo comprenderlo y aprender a manejarlo aumentando la velocidad de desarrollo para poder cumplir el apretado plazo de dos meses. Siendo cadenas de caracteres la mayoría de los datos a manejar, resultan de gran ayuda todas las funcionalidades que este lenguaje ofrece al respecto. Además, las bibliotecas que dispone para el manejo de archivos de Microsoft Excel son muy completas.

3.1.2 Sistema de gestión de bases de datos

Al comienzo del desarrollo del programa, se decidió emplear MySQL ya que ofrece muchas funcionalidades. Inicialmente, se encontraron problemas a la hora de instalar la biblioteca que permitía el manejo de este tipo de bases de datos desde Python. Aparentemente se solucionaron al colocar la carpeta que contiene la biblioteca en el mismo directorio en el que se encontraba el fichero de Python que la usaba. Más tarde, se descubrió que algunas funciones de dicha biblioteca generaban errores en la ejecución. Tras las dificultades que supusieron dichos errores, se optó por cambiar a SQLite. El hecho de que Python ya trae la biblioteca *sqlite3* preinstalada facilitó el cambio de motor de bases de datos. La sencillez de dicha biblioteca y motor resultó ser una ventaja ya que, realmente, todo el potencial que ofrece MySQL no se estaba aprovechando y complicaba las cosas.

3.1.3 Biblioteca para el manejo de archivos de Microsoft Excel

En un principio se consideró usar la biblioteca *pandas* por su sencillez. Pronto se vio la necesidad de una biblioteca más potente; *openpyxl* fue la opción elegida. El objeto tipo *worksheet* que ofrece esta biblioteca contiene suficientes métodos y atributos como para solucionar la mayor parte de problemas que se plantean. Además, la posibilidad de identificar las celdas combinadas (clave para la interpretación de la sábana) fue determinante en la elección de *Openpyxl* frente a *pandas*.

3.2 Diseño de la aplicación

Para un correcto funcionamiento se decide dividir el programa en: extracción de datos, filtrado de datos, creación de la base de datos y creación del motor de búsqueda e interfaz de usuario (Figura 3-1). Este diseño de la aplicación en partes aporta las siguientes ventajas:

- En caso de un cambio en el formato de la sábana, solo hace falta modificar la extracción de datos (parser).
- Solo hace falta extraer los datos una vez por sábana introducida.
- Se puede mejorar la interfaz sin que esto afecte a la extracción de datos.

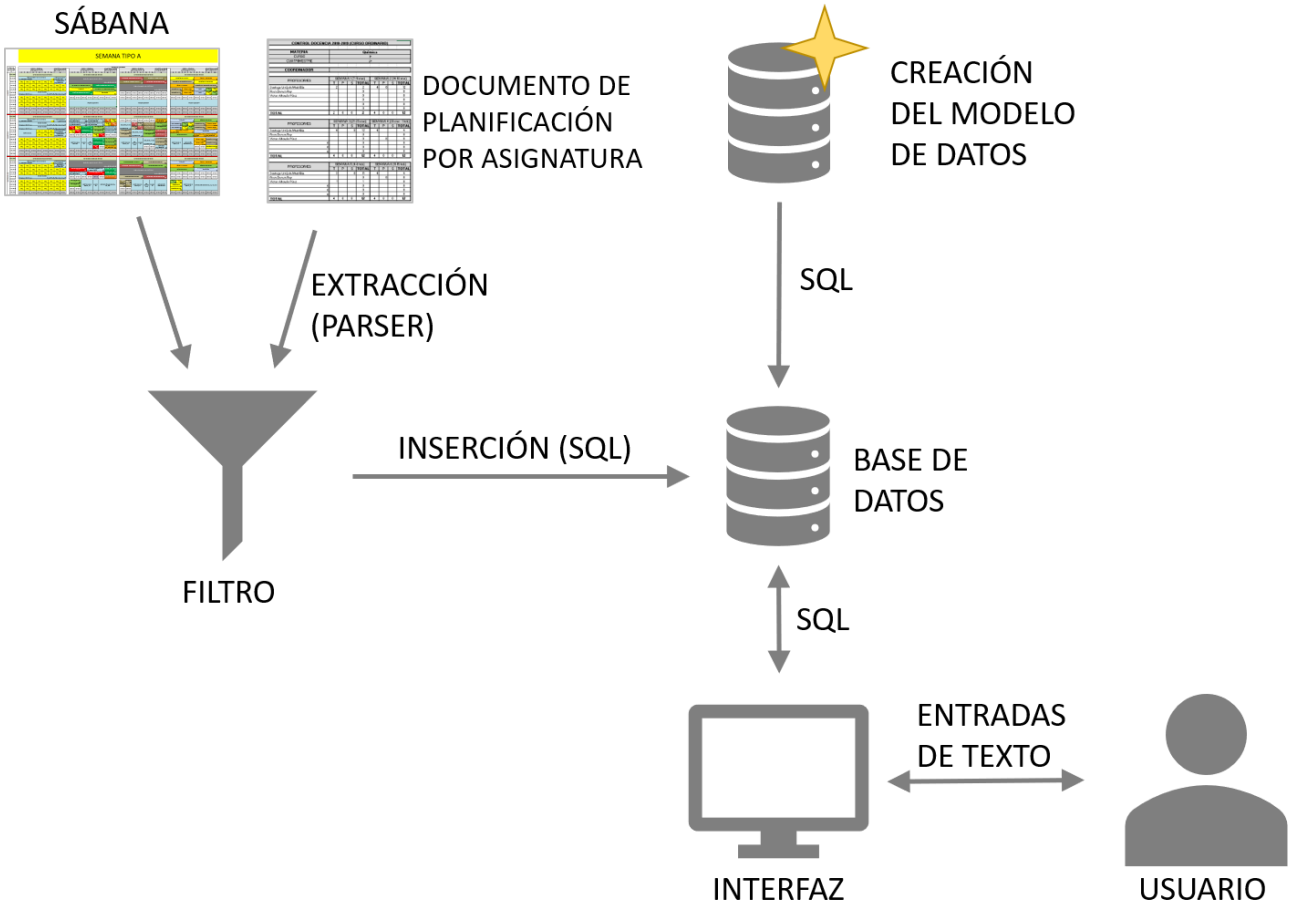


Figura 3-1 Diagrama del diseño de la aplicación

La base de datos relacional a crear se divide en cinco tablas: asignaturas, profesores, grupos de clase, aulas y sesiones. Todas las consultas que se realicen a través de la interfaz serán mediante comandos SQL sobre la base de datos.

3.2.1 Extracción de datos

Para rellenar las tablas mencionadas anteriormente se han obtenido los datos de diversas maneras:

- **Asignaturas**: se han obtenido de manera manual, es decir, se han copiado manualmente a una lista en la cual se les ha asociado un número identificador a cada una. Las asignaturas militares no han tenido en cuenta ya que se organizan de manera diferente.
- **Profesores**: se han obtenido a partir de una tabla que se encuentra en la página web del centro universitario. De esta manera cada vez que cambie el profesorado, se actualizará de manera automática en la base de datos siempre y cuando se disponga de conexión a Internet. En caso de no existir conexión, existe una lista de profesores almacenada con los profesores que había en el momento de la creación del programa. Esta tabla contiene los siguientes datos: identificador, apellidos, nombre, teléfono, rpv y correo electrónico.
- **Grupos de clase**: se obtienen a partir de la sábana que se esté usando, es decir, en la base de datos solo existirán los grupos de clase que aparezcan en la sábana existente. Para crear

cada grupo se combina el curso al que pertenece con el número de grupo y se le añade un número identificador.

- **Aulas**: se han obtenido de manera manual, es decir, se han copiado manualmente a una lista que contiene los nombres de las aulas junto con el número de sillas que hay en ellas y si se trata de un aula de ordenadores o no. También se les ha añadido su correspondiente número identificador.
- **Sesiones**: en el proceso de extracción de las sesiones están involucrados tanto la sábana como los archivos de planificación de las asignaturas. Si no se dispone del documento de planificación de una asignatura determinada, el campo de profesor quedará vacío. Es el proceso más complejo y que más recursos consume. Cada sesión consta de un número identificador, fecha, hora, asignatura, profesor, grupo y aula.

3.2.2 Filtrado de datos

Este proceso se encarga de asegurar que los datos extraídos se procesen de manera correcta. Solo afecta a los datos que no se obtienen de manera manual. El filtrado de profesores consiste únicamente en eliminar los acentos de nombres y apellidos. En cuanto a los grupos, el filtrado elimina la parte del nombre que contiene el número de promoción ya que éste no es relevante.

El proceso de filtrado de sesiones es el más laborioso ya que tiene que lidiar con las asignaturas militares. El primer paso consiste en diferenciar asignaturas militares de las del CUD. Una vez clasificada como asignatura del CUD, se sustituye el nombre de la asignatura por su correspondiente número identificador. Lo mismo ocurre con el profesor, el grupo y el aula.

3.2.3 Modelo de datos

Para diseñar la base de datos se utilizó la interfaz gráfica del programa MySQL. Tras crear la estructura de las tablas como se puede ver en la Figura 3-2, MySQL generaba un código en lenguaje SQL que, transcrito al fichero de Python, permitía crear la base de datos relacional desde el programa. Cuando se decide cambiar a SQLite, se realizan unas pequeñas modificaciones para adaptarlo manteniendo la estructura inicial.

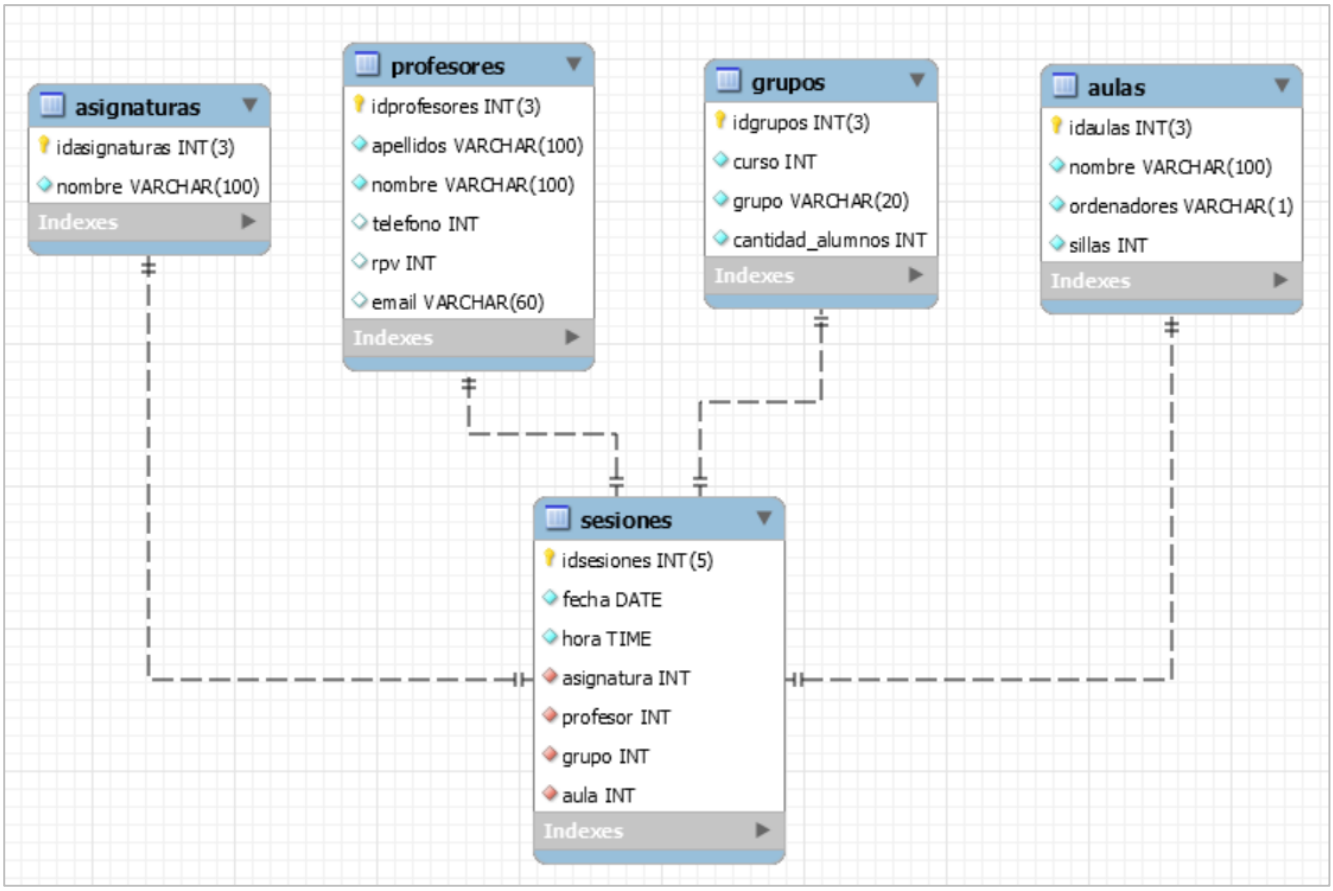


Figura 3-2 Modelo de datos

En el mismo fichero de Python que se encarga de generar la base de datos se encuentra el código necesario para insertar los datos, ya filtrados, en sus tablas correspondientes. De esta manera, la base de datos queda lista para ser objeto de consultas y modificaciones.

3.2.4 Creación del motor de búsqueda e interfaz de usuario

Con la base de datos creada y rellena se pueden realizar todo tipo de consultas y modificaciones a través de comandos SQL. El objetivo de la interfaz es que el usuario no tenga que conocer el lenguaje SQL para realizar dichas consultas o modificaciones. La interfaz se divide en cuatro partes principales:

- **Actualización de datos**: en esta parte se ejecuta el programa entero de manera que se extraen, filtran e insertan los datos desde cero. Esto sirve para cuando se quiere usar una sábana nueva o ha habido algún gran cambio que afecte a varias tablas.
- **Consultas**: aquí el usuario puede realizar diferentes consultas según le interese saber las sesiones de una asignatura, el horario de un profesor, el horario de un grupo de clase o las sesiones que tienen lugar en un aula.
- **Modificaciones**: esta parte está pensada para hacer pequeñas modificaciones en cualquiera de las cinco tablas que hay en la base de datos. Permite añadir datos, modificarlos o incluso eliminarlos.
- **SQL**: los usuarios avanzados que conozcan el lenguaje SQL podrá hacer uso de él en esta parte de la interfaz.

3.3 Implementación de la aplicación

En este apartado se describirán en detalle los procesos que realiza el código que hay escrito en los diferentes ficheros que conforman el programa. Para describirlos, se seguirá el orden cronológico de creación. El primero en realizarse fue *excel.py*, sin embargo, antes de finalizar su desarrollo, se

comenzó con *sqlite.py* de manera paralela. En último lugar se creó la interfaz de usuario en *consultas.py*.

Todos los ficheros de Python utilizados están encabezados por la declaración del uso de codificación UTF-8.

3.3.1 Extracción de datos

El fichero encargado de extraer los datos de la sábana y del documento de planificación de asignatura (Anexo I: Fichero excel.py~~Anexo III: Fichero excel.py~~) es el más complejo de todos. En él se definen las funciones necesarias para extraer ordenadamente la información necesaria de cada documento.

3.3.1.1 Bibliotecas utilizadas

En la primera parte del fichero se importan las bibliotecas que se van a necesitar en sus procesos. De la biblioteca *openpyxl* se importa la clase *load_workbook* con la que se lee la sábana y la clase *worksheet* utilizada para detectar las celdas combinadas. También se importa la función *listdir* de la biblioteca *os* para navegar por los directorios de la aplicación y las clases *date*, *timedelta* y *datetime* de la biblioteca *datetime* para hacer operaciones con fechas.

3.3.1.2 Carga de documentos

Tras importar las bibliotecas correspondientes, se recorre el directorio raíz de la aplicación en búsqueda de un archivo en formato Microsoft Excel que será la sábana con la que trate el programa. Dentro de este documento se buscan las hojas que corresponden a las semanas tipo A y tipo B, asignándolas a variables. Por último, se carga el primer documento de la carpeta de “Estimaciones” y se asigna su primera hoja a una variable con el fin de aprovechar su estructura en la identificación de semanas.

3.3.1.3 Definición de funciones

A continuación, se definen todas las funciones necesarias en el proceso de extracción de datos. También se definen las listas locales de asignaturas y aulas. Para entender el uso que tiene cada función en el proceso, se explicarán de más genéricas a más concretas. Clasificándolas en: funciones de primer orden o generadoras de listas, funciones de segundo orden y funciones de tercer orden.

3.3.1.4 Funciones generadoras de listas

Estas funciones se encuentran definidas en la última parte del fichero después de las listas de asignaturas y aulas.

3.3.1.4.1 Asignaturas

La primera que se define, *tabla_asignaturas*, es la que genera la lista para la tabla de asignaturas. El proceso es muy sencillo y consiste en recorrer la lista de asignaturas anexándolas a una lista junto a su correspondiente número identificador.

3.3.1.4.2 Profesores

La función utilizada para obtener la información referente al profesorado, *tabla_profs*, es la que tiene el funcionamiento más complejo. Como la información se adquiere de Internet, es necesario importar unas bibliotecas específicas para este fin. A pesar de que existen bibliotecas muy avanzadas para el manejo de documentos web, se optó por *urllib.request* y *html.parser* por ser parte de las que Python trae por defecto.

El archivo en formato HTML se descarga de Internet y se introduce en el parser de manera que filtre el lenguaje HTML y se obtenga como resultado el texto contenido en la página. El parser no procesa los acentos ni la letra "ñ" por tanto se utiliza una función para convertir los caracteres Unicode a la letra que corresponde.

Por último los datos se filtran y se anexan a una lista de manera ordenada con sus correspondientes números identificadores.

3.3.1.4.3 Grupos

Para generar la lista de grupos la función *tabla_grupos* recorre la fila 8 de una de las hojas de la sábana. En esta parte se encuentra una celda por cada grupo existente. Aplicando las funciones *grupo* y *brigada* se obtienen por separado el nombre del grupo y el curso correspondiente. Éstos se anexan a una lista junto a su número identificador correspondiente.

3.3.1.4.4 Aulas

El funcionamiento de esta función, *tabla_aulas*, es similar al de la función *tabla_asignaturas*. La única operación que realiza es insertar en cada elemento de la lista local de aulas su correspondiente número identificador.

3.3.1.4.5 Sesiones

La función *tabla_sesiones* es la que más recursos consume en su ejecución. En ella se usan la mayoría de las funciones de orden inferior. Para su correcto funcionamiento necesita de tres argumentos de entrada: la fecha en la que comienza el cuatrimestre, la primera fecha del intervalo a actualizar y la última.

Consiste en un bucle que recorre todas las fechas del intervalo comprendido entre los dos últimos argumentos cuyas fechas también se incluyen en el intervalo. Ya que para cada día que procesa tarda unos 30 segundos se vio necesario mostrar por pantalla el día que está procesando en ese instante para que el usuario sepa que la aplicación no se ha congelado. Antes de comenzar la inserción de datos, la función comprueba la semana del curso escolar para evitar procesar semanas no lectivas, si es tipo A o B y el día de la semana que es para no procesar sábados y domingos.

Una vez obtenidos dichos datos, se procede a recorrer todas las celdas proporcionadas por la función *tabla_dia*. Por cada celda individual se comprueba su pertenencia a una combinación de celdas con la función *madre*. Ninguna celda recorrida debe ser "huérfana", es decir, no puede haber ninguna celda vacía que no pertenezca a ninguna combinación de celdas. Por último, a través de las funciones *hora*, *asignatura*, *profesor*, *brigada*, *grupo* y *aula* se rellenan los campos de cada sesión con su correspondiente número identificador.

3.3.1.5 Funciones de segundo orden

Por funciones de segundo orden se entienden las funciones que tienen un uso directo en las funciones generadores de listas.

3.3.1.5.1 Función madre

Siendo una de las funciones con menos líneas de código, es de las que más recursos consume en la ejecución. Necesita de los siguientes argumentos: el objeto tipo celda y el objeto tipo *worksheet* del tipo de semana correspondiente. Su funcionamiento consiste en recorrer la lista de celdas combinadas generada por el atributo *merged_cells.ranges* del objeto tipo *worksheet*. Si la celda que entra como argumentos se encuentra dentro del rango de funciones que comprende cada combinación de celdas, devuelve la celda superior izquierda de dicha combinación.

Esta función es muy relevante para el correcto funcionamiento de la extracción de datos ya que en una celda combinada, la información solo se almacena en la celda superior izquierda de todas las que forman la composición.

3.3.1.5.2 Función hora

Recibiendo como argumentos el objeto tipo celda y el objeto tipo *worksheet* del tipo de semana correspondiente, lo único que hace esta función es devolver el valor que se encuentra en la columna 2 de la misma fila, es decir, la hora correspondiente a la sesión en cuestión.

3.3.1.5.3 Función asignatura

Esta función recibe un objeto de tipo celda como argumento y analiza su valor. Todos los nombres posibles que puede tener una asignatura militar se encuentran en la lista *militares*. Si el valor de la celda coincide con alguno de dicha lista la función devuelve el valor sin modificar.

En el caso de no tratarse de una asignatura militar, a través del método *split* y el método *pop* se elimina la última parte del nombre, es decir, se elimina la parte que contiene los grupos (Figura 3-3). De esta manera resulta más sencillo asociar el nombre de la signatura con el contenido de la celda. Una vez hecha la asociación, la función devuelve el nombre completo de la asignatura.

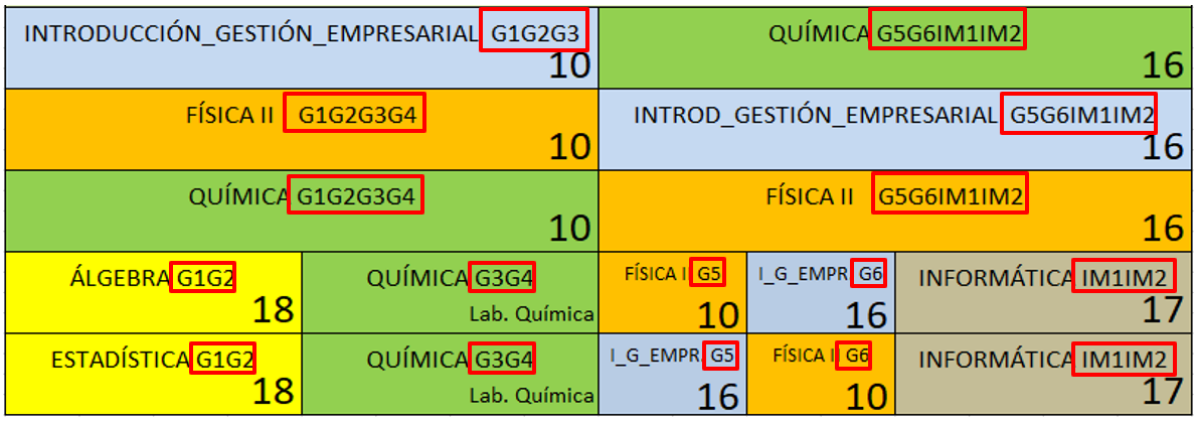


Figura 3-3 Grupos de clase con los nombres de las asignaturas

3.3.1.5.4 Función profesor

En este caso la función recibe tres argumentos: el objeto tipo celda, el número de la semana de curso y el objeto tipo *worksheet* de la hoja del tipo de semana correspondiente. Es la función de segundo orden más compleja y de las que más recursos consume en su ejecución.

Haciendo uso de la función *asignatura* comprueba si se trata de una asignatura militar. En caso afirmativo, busca el nombre del profesor en la parte inferior de la celda ya que en las asignaturas militares suele figurar (Figura 3-4). En caso de no ser militar, utiliza la función *excel_plan* para encontrar el documento de planificación de dicha asignatura en la carpeta “Estimaciones”. En caso de existir dicho documento, comprueba si se trata de una sesión de teoría, práctica o seminario para buscar al profesor al que le corresponda dicha clase en la semana indicada como argumento.

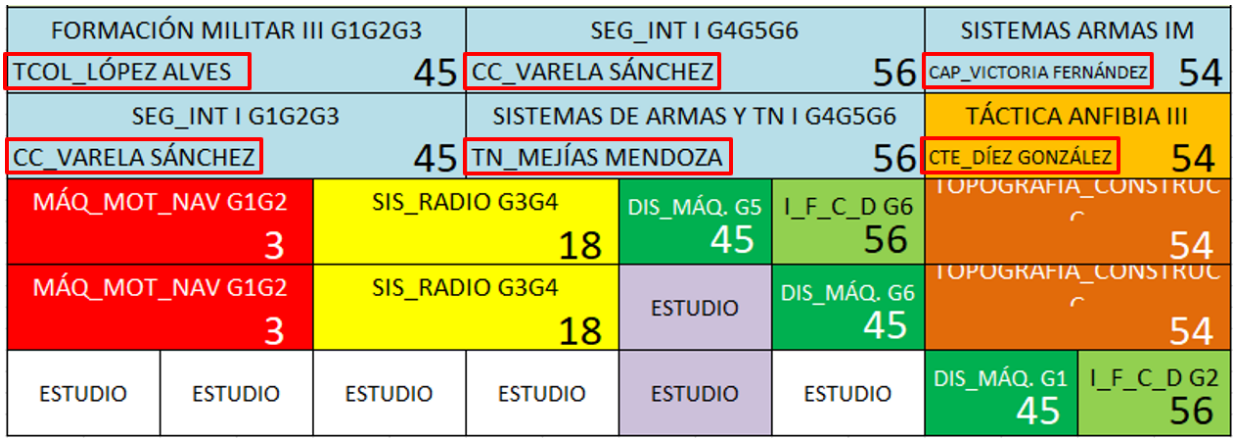


Figura 3-4 Nombres de los profesores militares en la parte inferior de las celdas

3.3.1.5.5 Función brigada

Esta función solo necesita como argumentos el objeto tipo celda y el objeto tipo *worksheet* del tipo de semana correspondiente. Su funcionamiento es muy sencillo y consiste en analizar el valor de la celda de la fila 7 que se encuentre en la misma columna. Se compara los diferentes empleos con dicho valor y el que coincida es el curso que devuelve.

3.3.1.5.6 Función grupo

Igual que en la función brigada esta función solo necesita como argumentos el objeto tipo celda y el objeto tipo *worksheet* del tipo de semana correspondiente. Su funcionamiento es igual de sencillo y consiste en devolver el valor de la celda de la fila 8 que se encuentre en la misma columna. Si no encuentra el grupo ahí, lo busca en la fila 7. En caso de que el nombre del grupo contenga el número de la promoción, se elimina esta parte.

3.3.1.5.7 Función aula

Utilizando como únicos argumentos el objeto tipo celda y el objeto tipo *worksheet* del tipo de semana correspondiente, esta función busca el aula en el que tiene lugar la sesión en diferentes partes de la celda. Para ello se ayuda de las funciones *alto* y *ancho* que devuelven las dimensiones de la celda en cuestión. En la mayoría de casos el aula se encuentra reflejada en la esquina inferior derecha de la celda, sin embargo también se puede encontrar en la parte inferior (Figura 3-5). Tras encontrar la posición, la función devuelve el aula.

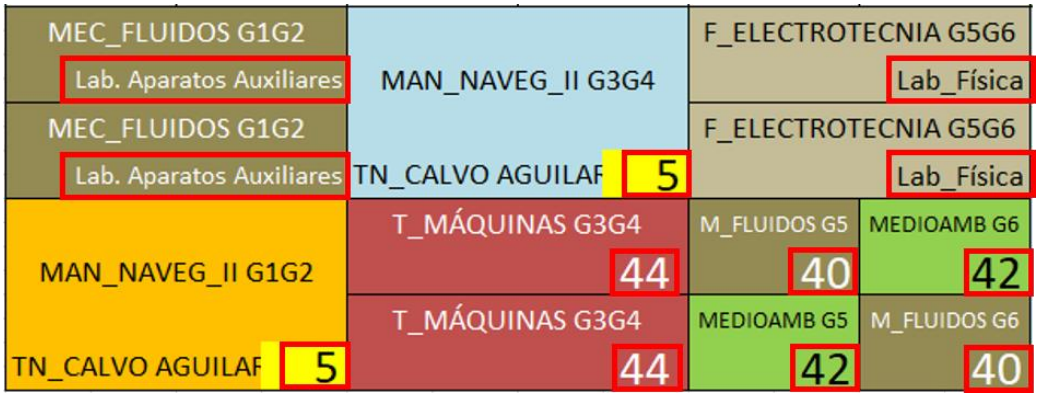


Figura 3-5 Posiciones del aula en las celdas

3.3.1.5.8 Función semana

Esta función recibe como argumentos una fecha en formato objeto tipo *date* y otro objeto similar con la fecha de comienzo de cuatrimestre. Basándose en un documento de planificación de asignatura,

calcula la semana de curso que corresponde a la fecha dada y la devuelve. En caso de tratarse de una semana no lectiva lo devuelve explícitamente.

3.3.1.5.9 Función tipo_semana

Recibiendo como único argumento el número correspondiente a una semana de curso y ayudándose de un documento de planificación de asignatura, determina si la semana en cuestión es tipo A o tipo B. Para ello tiene en cuenta que en las semanas de instrucción y adiestramiento (I+A) se repite un tipo de semana. Si la semana es no lectiva también lo devuelve explícitamente.

3.3.1.5.10 Función tabla_dia

Esta función necesita dos argumentos: el día de la semana en número (siendo el lunes 0 y el domingo 6) y el objeto tipo *worksheet* de la semana correspondiente. La lista que genera esta función es clave para el desarrollo de la lista de sesiones generada por *tabla_sesiones*. La función recorre toda la parte superior almacenando en una lista la columna en la que se encuentra cada grupo y el lateral derecho almacenando en otra lista la fila en la que se encuentra cada hora. Una vez recorridas ambas partes, cruza las dos listas de manera que se genera una nueva lista con todas las combinaciones de filas y columnas. Esta lista es lo que devuelve esta función y sirve de referencia para la función *tabla_sesiones* en cuanto a celdas a recorrer.

3.3.1.6 Funciones de tercer orden

Las funciones descritas en este apartado fueron creadas para realizar tareas sencillas y concretas necesarias en el desarrollo de funciones superiores.

3.3.1.6.1 Funciones alto y ancho

Teniendo en cuenta que las celdas de la sábana son realmente combinaciones de celdas, es necesario saber dónde comienzan y terminan dichas celdas. Después de colorear y agrupar las celdas, la sábana es fácil de interpretar al ojo humano (Figura 3-6). No ocurre lo mismo a la hora de interpretar la sábana mediante un programa. Este problema supuso grandes dificultades al comienzo del desarrollo de la aplicación. Las soluciones que se fueron probando se exponen a continuación.

CURSO 2018-2019 SEGUNDO CUATRIMESTRE		CUERPO GENERAL						INFANTERÍA MARINA
		ASP2 (CGA) PROM. 422						ASP2 (CIM) PROM. 152
		G1	G2	G3	G4	G5	G6	IM
ARTES	1050-1140	MAN_NAVEG_II G1G2		T_MÁQ G1 40	ELECTROT. G4 42	MEC_FLUIDOS G5G6 Lab. Aparatos Auxiliares		T_MEDIOAMBIENTAL IM Lab. Química
	1145-1235	TN_CALVO AGUILAR 5		ELECTROT. G3 42	T_MÁQ G1 40	MEC_FLUIDOS G5G6 Lab. Aparatos Auxiliares		T_MEDIOAMBIENTAL IM Lab. Química
	1240-1330	F_ELECTROTECNIA G1G2 Lab_Física		MAN_NAVEG_II G3G4		T_MEDIOAMBIENTAL IM Lab. Química		T_MÁQUINAS_MEC IM 44
	1335-1425	F_ELECTROTECNIA G1G2 Lab_Física		TN_CALVO AGUILAR 5		T_MEDIOAMBIENTAL IM Lab. Química		T_MÁQUINAS_MEC IM 44

Figura 3-6 Sábana vista al ojo humano

Usando las funciones básicas de la librería openpyxl, el código interpretaba la sábana como se puede ver en la Figura 3-7. Por ello la única solución era contar las celdas vacías que hay entre las llenas. Esto funcionaba en la mayoría de los casos, pero fallaba cuando la celda de al lado era más alta o más ancha.

CURSO 2018-																
		CUERPO GENERAL												INFANTERÍA MARINA		
		ASP2 (CGA) PROM. 422												ASP2 (CIM) PROM. 152		
		G1		G2		G3		G4		G5		G6		IM		
	1050-1140	MAN_NA				T_MÁQ		ELECTROT		MEC_FLUI				T_MEDIO		
							40		42	Lab.				Lab.		
	1145-1235					ELECTROT		T_MÁQ		MEC_FLUI				T_MEDIO		
		TN_CALV			5		42		40	Lab.				Lab.		
	1240-1330	F_ELECTR				MAN_NA				T_MEDIO				T_MÁQUI		
		Lab_Física								Lab.						44
	1335-1425	F_ELECTR								T_MEDIO				T_MÁQUI		
		Lab_Física				TN_CALV			5	Lab.						44

Figura 3-7 Sábana interpretada por el código con funciones básicas

Al ver la poca fiabilidad del método anterior, se vio la necesidad de encontrar otro método que permitiera definir los límites de las celdas de manera precisa. Se investigó por tanto los parámetros referentes al formato y color de las celdas buscando el objetivo que muestra la Figura 3-8. Sin embargo, los atributos de los objetos referentes al formato eran inconsistentes a la hora de definir el color de fondo. Tras múltiples pruebas se decidió que este método era incluso menos fiable que el anterior.

CURSO 2018-																
		CUERPO GENERAL												INFANTERÍA		
		ASP2 (CGA)												ASP2 (CIM)		
		G1		G2		G3		G4		G5		G6		IM		
	1050-1140	MAN_NA				T_MÁQ		ELECTROT		MEC_FLUI				T_MEDIO		
							40		42	Lab.				Lab.		
	1145-1235					ELECTROT		T_MÁQ		MEC_FLUI				T_MEDIO		
		TN_CALVO AGUILAR			5		42		40	Lab.				Lab.		
	1240-1330	F_ELECTR				MAN_NA				T_MEDIO				T_MÁQUI		
		Lab_Física								Lab.						44
	1335-1425	F_ELECTR								T_MEDIO				T_MÁQUI		
		Lab_Física				TN_CALVO AGUILAR			5	Lab.						44

Figura 3-8 Sábana interpretada por funciones capaces de diferenciar colores

La investigación sobre los objetos y las funciones disponibles en la biblioteca *openpyxl* se retomó y se dio con la solución definitiva. Existe un atributo del objeto tipo *worksheet* que lista todas las celdas combinadas (Figura 3-9). Comparando las celdas con dicha lista los resultados son fiables, sin embargo, recorrer la lista de celdas combinadas por cada celda que se quiere comprobar consume bastantes recursos.

CURSO 2018-2019 SEGUNDO CUATRIMESTRE		CUERPO GENERAL						INFANTERÍA MARINA
		ASP2 (CGA) PROM. 422						ASP2 (CIM) PROM. 152
		G1	G2	G3	G4	G5	G6	IM
MARTES	1050-1140	MAN_NAVEG_II G1G2		T_MÁQ G1	ELECTROT. G4	MEC_FLUIDOS G5G6		T_MEDIOAMBIENTAL IM
				40	42	Lab. Aparatos Auxiliares		Lab. Química
	1145-1235			ELECTROT. G3	T_MÁQ G1	MEC_FLUIDOS G5G6		T_MEDIOAMBIENTAL IM
		TN_CALVO AGUILAR 5		42	40	Lab. Aparatos Auxiliares		Lab. Química
	1240-1330	F_ELECTROTECNIA G1G2		MAN_NAVEG_II G3G4		T_MEDIOAMBIENTAL IM		T_MÁQUINAS_MEC IM
		Lab_Física				Lab. Química		44
	1335-1425	F_ELECTROTECNIA G1G2				T_MEDIOAMBIENTAL IM		T_MÁQUINAS_MEC IM
		Lab_Física		TN_CALVO AGUILAR 5		Lab. Química		44

Figura 3-9 Sábana interpretada a través de la lista de celdas combinadas

El funcionamiento de las funciones *alto* y *ancho* se basa en dicha lista. Necesitan como argumentos el objeto tipo celda y el objeto tipo *worksheet* del tipo de semana correspondiente. Con esto devuelven el número de celdas verticalmente y horizontalmente respectivamente.

3.3.1.6.2 Función columna

A pesar de que existe un atributo del objeto tipo celda que devuelve la columna, este atributo se refiere a la letra o letras que representan la columna. Para ciertas partes del código se prefiere que la columna se reciba de manera numérica ya que es más fácil de operar así. Esta función se encarga de traducir las letras al número correspondiente usando un alfabeto.

3.3.1.6.3 Función excel_plan

Recibiendo como único argumento el nombre de una asignatura, esta función se encarga de devolver el nombre del archivo que contiene la planificación de dicha asignatura. Esta función clave en desarrollo de la función *profesor*. Si no existe el archivo buscado la función lo explicita.

3.3.1.6.4 Función plan_semana

Esta función tiene como argumentos una semana y el nombre de un documento de planificación de asignatura. Con esta información devuelve la parte del documento donde se encuentran distribuidas las sesiones de la semana en cuestión.

3.3.1.6.5 Funciones teoria, practica y seminario

Estas funciones son necesarias a la hora de determinar el profesor al que le corresponde impartir una sesión. Todas reciben como argumentos el objeto tipo celda y el objeto tipo *worksheet* del tipo de semana correspondiente. Analizando las dimensiones de las celdas son capaces de hacer la discriminación. La lógica es la siguiente: si son dos sesiones iguales seguidas es una práctica; si tiene 2 o 4 celdas de ancho y no es una práctica, entonces es un seminario; en el resto de casos se trata de una sesión de teoría. En la función *seminario* hay consideradas más condiciones para identificar un seminario.

3.3.1.6.6 Funciones dia_semana y nombre_dia

Ambas funciones están fuertemente relacionadas ya que se utilizan en la función *tabla_dia* para comprobar que el día que recibe dicha función como argumento es el mismo que el de las filas con las que se crea la lista resultante.

La función *nombre_dia* solo sirve como traductor de número a día de la semana en letra (como aparece en la sábana). La función *día_semana* por otro lado, utiliza como argumentos el objeto tipo celda y el objeto tipo *worksheet* del tipo de semana correspondiente para devolver el día de la semana que aparezca en la columna 2 siguiendo la fila de la celda en cuestión.

3.3.1.6.7 Función normalizar

Tomando como argumento una cadena de caracteres cualquiera, la función quita todos los acentos que aparezcan y convierte todas las letras en minúsculas a excepción de la primera de la cadena y los números romanos que pueda haber al final, por ejemplo, "FÍSICA II" pasaría a ser "Fisica II".

3.3.1.7 Listas locales

En el fichero existen tres listas locales: *asignaturas*, *militares* y *lista_aulas*. Para el funcionamiento de muchas de las funciones, se comparan los valores de las celdas con las listas de asignaturas del CUD y asignaturas militares. De esta manera se pueden identificar y procesar correctamente. La lista de aulas, sin embargo, solo se utiliza en la función *tabla_aulas* para generar la lista con la que se crea la tabla aulas de la base de datos.

3.3.1 Creación de la base de datos

El fichero encargado de la creación de la base de datos (Anexo II: Fichero sqlite.py) es el que se ejecuta a través de la primera opción del menú principal de la interfaz de usuario. En este fichero se crea llama a las funciones que extraen los datos de los documentos usados. En el caso de realizarse la extracción con éxito, se procede a crear la estructura de la base de datos y a rellenarla con dichos datos.

3.3.1.1 Bibliotecas importadas

El fichero comienza importando la biblioteca *sqlite3*, necesaria para la creación de la base de datos y la inserción de datos en la misma. También se importan las clases *datetime* y *timedelta* de la biblioteca *datetime* con el único propósito de medir el tiempo de que necesita el fichero para ejecutarse. Se importa la función *load_workbook* de la biblioteca *openpyxl* necesaria para el argumento de la función que genera la lista de grupos de clase. Se importan del fichero de extracción las funciones encargadas de crear las listas con las que se rellena la base de datos. También se importa la función *system* de la biblioteca *os* para poder limpiar el terminal antes de empezar el proceso.

3.3.1.2 Introducción de fechas necesarias

Comienza con un bucle en el cual, tras limpiar la consola, se solicita al usuario que indique la fecha en la que comienza el cuatrimestre y el intervalo de fechas objeto de la actualización. En el caso de introducirse un cero se reinicia el bucle. Una vez introducidos todos los números necesarios para componer las fechas, se comprueba a través de un control de excepciones que los números introducidos forman una fecha lógica. Si el resultado es positivo se rompe el bucle dando paso a la extracción de datos. En caso contrario, se solicitan de nuevo las fechas.

3.3.1.3 Creación de las listas de datos

Justo antes de empezar la extracción de datos, se almacena la hora en una variable para su posterior uso. Para la extracción de datos se ejecutan las funciones importadas encargadas de dicha labor y se almacenan las listas resultantes en variables. La única de estas funciones que precisa de argumento es la de *tabla_sesiones* ya que su alcance depende de las fechas introducidas anteriormente.

3.3.1.4 Filtrado de datos

Una vez extraídos los datos de sus fuentes correspondientes, se procede al filtrado de los mismos utilizando un bucle *for* para recorrer la información que se encuentra en la lista de que corresponde a la

tabla de sesiones. Cuando se encuentra una coincidencia de nombres de asignatura, profesor, grupo de clase o aula, se sustituye por su correspondiente número identificador.

3.3.1.5 Modelo de datos

A continuación, se conecta con la base de datos (si no existiese, este comando se encargaría de crearla) y se crea el cursor. Usando el método *execute* se crean las tablas de manera secuencial con el comando SQL *CREATE TABLE IF NOT EXISTS*. Si se estuviese sobrescribiendo la base de datos, las tablas serían eliminadas y creadas de nuevo para evitar posibles errores.

3.3.1.6 Inserción de datos en las tablas

Para completar la base de datos, se recorren las listas que contienen los datos extraídos anteriormente y se insertan en sus respectivas tablas haciendo uso del método *execute* y el comando SQL *INSERT*.

Por último, se cierra la base de datos ya completa y se calcula el tiempo que ha llevado el proceso mostrándoselo por la pantalla al usuario.

3.3.2 Consultas

El fichero que contiene el código correspondiente a la interfaz (Anexo III: Fichero consultas.py~~Anexo I: Fichero consultas.py~~) comienza con la importación de la función *system* de la biblioteca *os*. Dicha función permite al programa ejecutar comandos propios de la línea de comandos como limpiar la pantalla, que es precisamente el objetivo de dicha importación.

La interfaz limpia la consola y da la bienvenida al usuario entrando en un bucle en el que ofrece cinco opciones: actualizar base de datos, hacer una consulta, modificar base de datos, SQL y salir. A través de una entrada de texto el usuario elije una opción.

3.3.2.1 Actualizar base de datos

Eligiendo la primera opción, se vuelve a limpiar el terminal y se ejecuta el fichero desde el que realiza la actualización (Anexo II: Fichero sqlite.py). Una vez finalizada, el programa espera una entrada de texto antes de volver al bucle inicial.

3.3.2.2 Hacer una consulta

Eligiendo la segunda opción, se entra en un nuevo bucle en el cual se importa la biblioteca *sqlite3* que permite el manejo de la base de datos. Usando funciones de dicha biblioteca se conecta con la base de datos y se crea un cursor, luego se limpia el terminal. Se solicita al usuario que indique la fecha en la que comienza el cuatrimestre y el intervalo de tiempo objeto de consulta mediante entradas de texto. En el caso de que el usuario introduzca un cero, el bucle se ropería volviendo al bucle inicial.

Una vez introducidas las fechas, se entra en un nuevo bucle en el cual se limpia la consola y se ofrecen al usuario los tipos de consultas. A través de una entrada de texto el usuario introduce su elección y se limpia el terminal de nuevo. Todas las opciones funcionan de manera similar a excepción del cero que rompería el bucle volviendo a la parte donde se introducen las fechas.

Tomando como ejemplo la consulta por asignaturas, se ejecuta un comando SQL a través del método *execute* del cursor que obtiene las asignaturas consultables. Con un bucle *for* y el método *fetchall* se imprimen en pantalla las asignaturas de manera ordenada gracias al método *format* de las cadenas de caracteres. Mediante una entrada de texto el usuario elige una asignatura siendo cero la manera de volver al bucle superior. Una vez elegida la asignatura, se limpia el terminal y se recurre de nuevo al método *execute* del cursor para obtener el resultado. El comando SQL utilizado en este caso es algo más complejo ya que usa información de todas las tablas de la base de datos. La consulta se realiza sobre la tabla de sesiones a través del número identificador de la asignatura seleccionada y

teniendo en cuenta las fechas introducidas anteriormente. Usando un *LEFT OUTER JOIN* se obtienen el nombre del profesor, el grupo de clase y el nombre del aula en lugar de sus números identificadores.

Usando de nuevo el método *fetchall*, se obtienen los resultados de la consulta y se almacenan en formato de lista de tuplas dentro de la variable *resultado*. Los resultados de la consulta se analizan para juntar las entradas de sesiones iguales con grupos diferentes, ya que se encuentran en la misma aula. Para poder realizar este análisis, se almacena la cantidad de tuplas en la variable *longitud* y se comienza a recorrer la lista usando un bucle *for*. Por cada tupla se almacena su posición en la variable *indice* y se recorren las tuplas que se encuentren después. Comparando fecha, hora, profesor y aula, si se detecta una coincidencia, se convierte la tupla inicial en lista para poder ser operada. A continuación, se añade el grupo a la tupla inicial y, tras convertirla de nuevo en tupla, se modifica la tupla analizada convirtiéndola a una tupla con un número de referencia y cadenas de caracteres "ERROR" y "CLASS". Una vez comprobadas todas las tuplas se procede a eliminar todos los elementos de *resultado* que sean clase lista, es decir, las que fueron modificadas con las cadenas "ERROR" y "CLASS".

Para presentar en la pantalla los resultados definitivos, se obtienen los títulos de las columnas consultadas haciendo uso del atributo del cursor *description*. Se imprimen en pantalla con ayuda del método *format* y, usando un bucle *for* y el método *format* de nuevo, se imprimen los resultados definitivos de la consulta. A continuación, se vuelven a analizar los resultados del mismo modo pero, esta vez, en búsqueda de posibles solapes. Si coinciden los profesores de dos tuplas y las aulas no o viceversa, se imprimirá por pantalla un mensaje de advertencia y las tuplas en conflicto. Por último, se espera a que el usuario introduzca una entrada de texto aleatoria para dejar de mostrar los resultados y terminar el bucle cerrando la conexión a la base de datos.

El resto de opciones de consulta funcionan igual con la excepción de la consulta de grupos que no necesita juntar los grupos de los resultados.

3.3.2.3 Modificar base de datos

Si en el menú principal se escoge la tercera opción, se entra en el bucle correspondiente a las modificaciones de la base de datos. Comienza importando la biblioteca *sqlite3*, conectándose a la base de datos y creando el cursor. Se limpia la consola para mostrar las cinco tablas que se pueden modificar: asignaturas, profesores, grupos, aulas y sesiones. Con una entrada de texto el usuario elige la tabla que desea modificar y se almacena en la variable *tabla*. En el caso de elegir la tabla de sesiones, se limpia el terminal para indicarle al usuario que debe introducir el día sobre el que va a realizar la modificación ya que si se muestra esta tabla entera se puede colapsar la consola. En el caso de introducirse un cero se rompe el bucle volviendo al menú principal.

Con la tabla ya seleccionada se limpia la consola para mostrar tres nuevas opciones: modificar entrada, añadir entrada y eliminar entrada. Con una entrada de texto el usuario hace su elección y se vuelve a limpiar el terminal. Se deja la primera línea vacía por motivos de presentación.

3.3.2.3.1 Modificar una entrada

En el caso de seleccionar la primera opción se comprueba si la tabla objeto de la modificación es la de sesiones, en ese caso, debe tener en cuenta la fecha que se introdujo. Usando el mismo procedimiento que en la parte de consultas, se obtienen los títulos de las columnas de la tabla en cuestión. Usando el método *fetchall* se obtienen las tuplas correspondientes a la tabla y se almacenan en la variable *resultado*. Teniendo en cuenta el número de columnas que tenga la tabla se genera un "esqueleto" para ser utilizado con el método *format* más tarde. Se imprimen en pantalla los títulos de las columnas haciendo uso del "esqueleto" y a continuación los datos de la tabla haciendo uso de un bucle *for* y el mismo "esqueleto". Ahora el usuario debe escoger qué entrada es la que quiere modificar.

En el caso de introducir un cero se obviaría el resto del bucle actual volviendo al inicio de este. Con la entrada a modificar elegida se utiliza un bucle *for* para mostrar las columnas modificables. Mediante una entrada de texto el usuario elige la columna que desea modificar y se la muestra solicitándole con una nueva entrada de texto que introduzca el valor deseado. Con toda la información necesaria para realizar la modificación se adquiere la tupla usando los métodos *execute* y *fetchone*. Se convierte en clase lista para poder ser editado, se edita y se vuelva a convertir en clase tupla. Por último, se inicia un bucle en el cual el usuario debe confirmar si la modificación mostrada es la deseada. De ser la respuesta afirmativa, la modificación se hace efectiva con el método *execute* y el comando SQL *UPDATE*. Se informa al usuario de que la modificación ha sido efectiva y se espera a que se introduzca algo antes de salir de este último bucle y pasar al cierre de conexión con la base de datos.

3.3.2.3.2 Añadir una entrada

Si se opta por añadir una entrada el proceso es más sencillo. Se obtiene el número identificador de la última entrada existente y se crea el "esqueleto" de la misma manera que en la parte de modificación. Recorriendo los títulos con un bucle *for* se va solicitando al usuario que vaya introduciendo los valores deseados. Con todos los valores almacenados en una lista, se inicia el bucle de comprobación donde se pregunta al usuario si esa es la entrada que quiere añadir a la tabla en cuestión. En caso afirmativo la inserción se hace efectiva con el método *execute* y el comando SQL *INSERT*. Se informa al usuario de que la entrada ha sido añadida y se espera a que se introduzca algo antes de salir de este último bucle y pasar al cierre de conexión con la base de datos.

3.3.2.3.3 Eliminar una entrada

Siendo la opción de eliminar entrada la elegida el procedimiento es todavía más sencillo. De manera idéntica que en la parte de modificaciones, se muestran al usuario todas las entradas de la tabla seleccionada y se solicita en este caso que elija cual quiere eliminar. Una vez elegida se procede directamente al bucle de comprobación.

3.3.2.4 SQL

La cuarta opción que nos ofrece el menú principal es SQL. Comienza limpiado el terminal e importando la biblioteca *sqlite3*. Se realiza la conexión a la base de datos y se crea el cursor. A continuación, comienza un bucle donde se indica al usuario que escriba el comando SQL que desee. Si se introduce un cero se rompe el bucle y se vuelve al menú principal. Para asegurar que el comando SQL está bien redactado, la ejecución del mismo se realiza bajo control de excepciones. Si el comando no es válido se le indicará al usuario y se volverá al inicio del bucle. Si el comando es un *SELECT*, es decir, si se espera una salida, se utilizará el método *fetchall* y se imprimirán por pantalla todos los resultados con los títulos de las columnas correspondientes. El "esqueleto" que se utilice para insertar los elementos de las tuplas será estándar y por tanto podrá darse el caso de que el dato sobrepase la dimensión esperada o que se desperdicie espacio. Por último, se guardan los posibles cambios realizados y se cierra la conexión con la base de datos.

3.3.2.5 Salir

Si el usuario elige la última opción del menú principal, es decir, introduce un 0, el bucle inicial se romperá y se terminará la ejecución del programa.

4 MANUAL DE USUARIO

4.1 Instalación de la herramienta

Para poder hacer uso de la herramienta necesita cumplir los siguientes requisitos:

- Tener un ordenador con el sistema operativo de Windows 7 o superior
- Tener instalada la última versión de Python
- Tener instalada la biblioteca *openpyxl* para Python
- Disponer de todos los ficheros que componen la herramienta
- Disponer de al menos una sábana en formato *.xlsx*
- Disponer de al menos un documento de planificación de asignatura en formato *.xlsx*

4.1.1 Instalación de Python

Para instalar Python todo lo que necesita es acceder a la página web oficial y descargar la versión que haya disponible para su sistema operativo en https://www.python.org/downloads/.

Para completar la instalación de Python ejecute y siga atentamente los pasos del instalador.

4.1.2 Instalación de Openpyxl

Una vez instalado Python, para instalar la biblioteca *openpyxl* debe acceder a la siguiente página: https://pypi.org/project/openpyxl/. En la página encontrará un link de descarga para descargar la biblioteca comprimida. Descomprima los archivos e inserte el fichero *instalar-openpyxl.bat* en la carpeta que contiene el fichero *setup.py*. Haga doble clic sobre el fichero *instalar-openpyxl.bat* y la biblioteca se instalará en unos instantes.

4.1.3 Ubicación de los archivos

Antes de arrancar la herramienta, asegúrese de que existe un archivo *.xlsx* de la sábana en el mismo directorio que los ficheros del programa. Compruebe también que existe al menos un documento de planificación de asignaturas y colóquelo en la carpeta “Estimaciones”.

4.2 Cómo usar la herramienta

Para iniciar la herramienta haga doble clic en el archivo *Iniciar.bat*. Debería visualizar el menú principal en una ventana de comandos como se muestra en la Figura 4-1. Se recomienda utilizar el programa en pantalla completa.

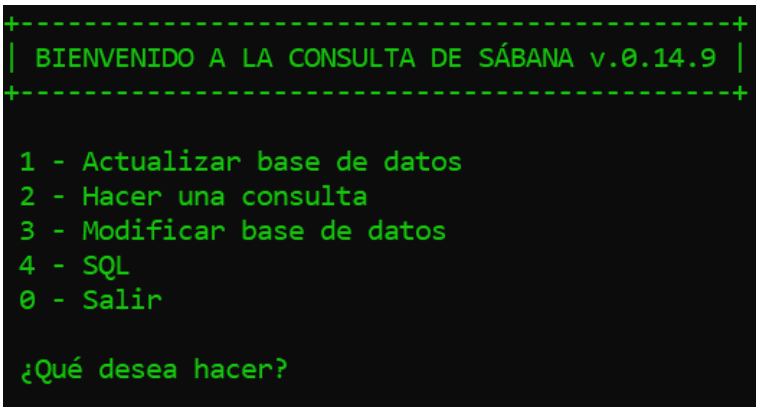


Figura 4-1 Menú principal

4.2.1 Menú principal

Para escoger una opción del menú, escriba el número correspondiente y pulse *enter*.

Si es la primera vez que ejecuta la herramienta y no dispone de un archivo *sabana.db* en el directorio de la herramienta, tendrá que seleccionar la primera opción: "Actualizar base de datos". De esta manera se creará la base de datos con la que trabajará la herramienta. Este proceso puede tardar alrededor de 40 minutos si se generan datos para un cuatrimestre completo.

Para salir del programa introduzca un 0.

4.2.2 Actualizar la base de datos

Esta opción debe ser seleccionada cada vez que, con motivo de cambios en la sábana o en el profesorado, se deseen actualizar los datos que contiene la base de datos. El proceso de actualizar los datos de todo un cuatrimestre requiere alrededor de 40 minutos para completarse.

Se recomienda indicar únicamente las fechas que vayan a ser objeto de consultas. Para volver al menú principal, introduzca un 0.

4.2.3 Hacer una consulta

Se debe introducir el intervalo de días que se desea consultar siguiendo estrictamente el formato indicado (AAAA-MM-DD). Por ejemplo: "2019-02-25".

A continuación, se ofrecen cuatro tipos de consulta: por asignatura, por profesor, por grupo de clase o por aula. Para escoger una opción, escriba el número correspondiente y pulse *enter*. En cualquiera de las opciones se mostrarán las sesiones relativas a la opción seleccionada.

En el caso de existir un solape, se indicará en pantalla.

Para volver al menú principal introduzca un 0.

4.2.4 Modificar base de datos

En primer lugar, debe indicar la tabla que desea modificar. En el caso de que sea la tabla de sesiones, debe indicar el día en el que se desea realizar la modificación.

Existen tres tipos de modificaciones: modificar entrada, añadir entrada y eliminar entrada.

Para modificar una entrada se mostrarán las entradas existentes, tras seleccionar la entrada que se desea modificar, deberá indicar la columna que desea modificar. Una vez introducido el nuevo valor, el programa le preguntará si la modificación que se desea realizar es la que aparece en pantalla. En caso afirmativo, introduzca un 1 y pulse *enter*.

Para añadir una entrada, escriba los valores correspondientes a las columnas en el orden en el que se solicitan. Al finalizar, el programa le preguntará si la información mostrada en pantalla es la que se desea introducir. En caso afirmativo, introduzca un 1 y pulse *enter*.

Para eliminar una entrada se mostrarán las entradas existentes. Tras seleccionar la entrada que se desea eliminar, el programa le preguntará si la información mostrada en pantalla es la que se desea eliminar. En caso afirmativo, introduzca un 1 y pulse *enter*.

Para regresar al menú principal introduzca un 0.

4.2.5 SQL

En esta parte del programa pueden introducir comandos SQL directamente en el terminal. Si no conoce dicho lenguaje, no se recomienda utilizarlo para modificar la base de datos ya que podría inutilizarla.

Si aparece el mensaje “Comando no válido.”, significa que la orden no se ha hecho efectiva.

Para volver al menú principal introduzca un 0.

4.2.6 Consejos

Usando el programa en pantalla completa los resultados de las consultas o modificaciones se mostrarán con más claridad.

En cualquier parte del programa se puede introducir un 0 para volver al submenú anterior o al menú principal.

Si cree que existe algún problema con la base de datos se recomienda actualizarla usando la primera opción del menú principal.

5 PRUEBAS Y VALIDACIÓN

Una vez completado el desarrollo de la aplicación, ésta se sometió a las siguientes pruebas para evaluar su funcionamiento.

5.1 Cambio de sábana

La aplicación está pensada para soportar sábanas futuras, sin embargo, para comprobar su eficacia, se han utilizado sábanas de años anteriores. Para que la aplicación sea capaz de extraer correctamente la información la sábana en cuestión debe tener la misma estructura que la actual. Esto se cumple para sábanas de años anteriores pero se detectó un problema relacionado con otro aspecto.

Para identificar correctamente las asignaturas y aulas, se utiliza una lista que contempla los posibles nombres de éstas. Si el nombre de la asignatura o aula que figura en la sábana no se encuentra en dicha lista, no se procesa correctamente. La inconsistencia de los nombres de asignaturas y aulas en las sábanas (Figura 5-1) hace que la única manera de solucionar este problema sea actualizar la lista de posibles nombres.

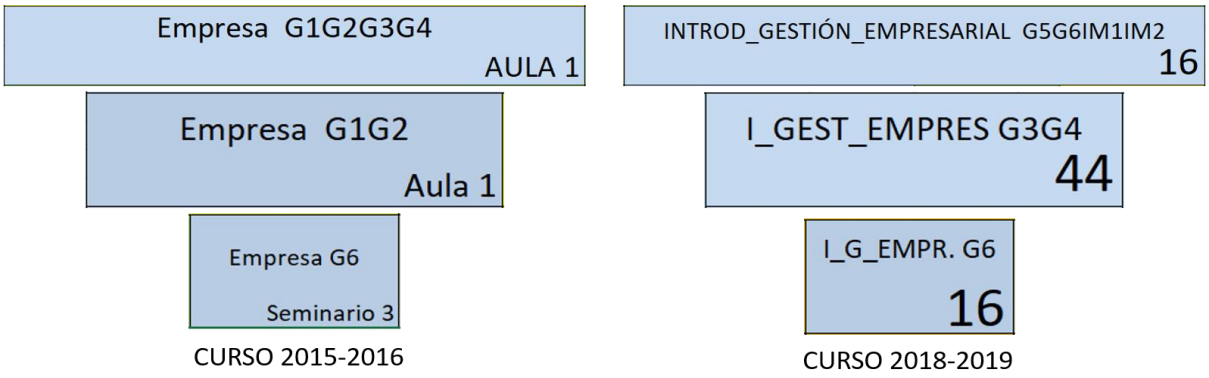
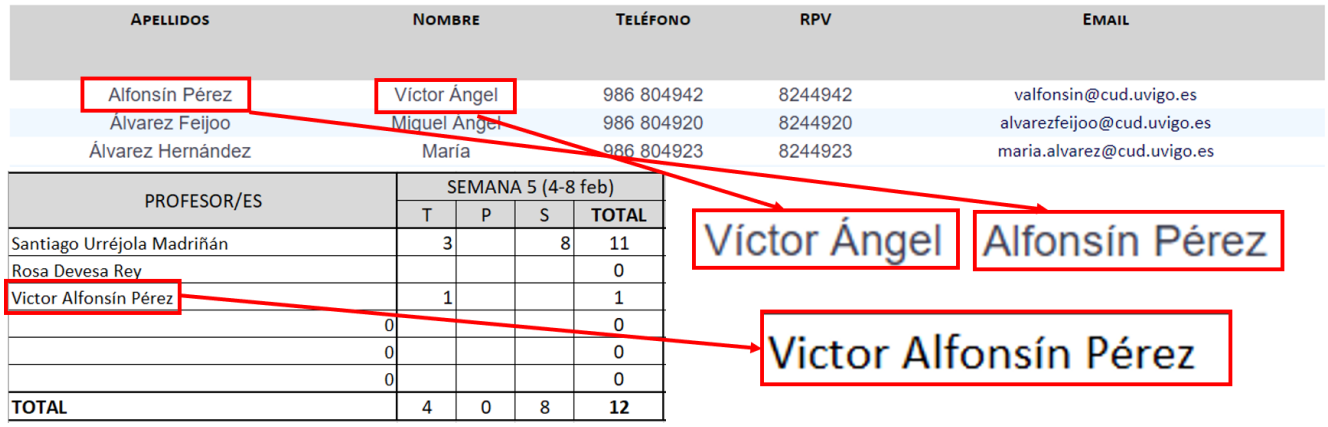


Figura 5-1 Ejemplo de los nombres de asignaturas y aulas en sábanas de diferentes cursos

5.2 Cambio de documento de planificación de asignatura

El formato de los documentos de planificación de asignatura es único y no presenta inconvenientes en ese aspecto. El único problema encontrado en esta prueba se producía cuando los nombres de los profesores no estaban completos o se encontraban escritos sin acentos y no se procesaban correctamente al compararlos con la tabla de profesores (Figura 5-2). Dicho problema se superó comparando solo los apellidos y eliminando los acentos de ambas partes.



APELLIDOS	NOMBRE	TELÉFONO	RPV	EMAIL
Alfonsín Pérez	Víctor Ángel	986 804942	8244942	valfonsin@cud.uvigo.es
Álvarez Feijoo	Miguel Ángel	986 804920	8244920	alvarezfeijoo@cud.uvigo.es
Álvarez Hernández	María	986 804923	8244923	maria.alvarez@cud.uvigo.es

PROFESOR/ES	SEMANA 5 (4-8 feb)			
	T	P	S	TOTAL
Santiago Urréjola Madriñán	3		8	11
Rosa Devesa Rey				0
Victor Alfonsín Pérez	1			1
0				0
0				0
0				0
TOTAL	4	0	8	12

Figura 5-2 Comparación de los nombres de la tabla de profesores con el documento de planificación

5.3 Creación de un cuatrimestre completo

La primera prueba que se realizó para comprobar la eficacia del sistema en este aspecto tuvo resultados positivos pero no satisfactorios ya que el proceso duraba cerca de dos horas. Tras revisar numerosas veces las operaciones realizadas, se consiguió reducir el tiempo a 30 minutos. Sin embargo, tras solucionar algunas irregularidades que se encontraron en otras partes de la aplicación el tiempo volvió a ascender, esta vez a 40 minutos. Cabe reseñar que el equipo utilizado dispone de un procesador Intel® Core™ i5-8250U y una memoria RAM DDR4 de 8GB.

La razón de este problema es la enorme cantidad de operaciones que tiene que hacer. En un cuatrimestre hay unos 80 días, en un día hay 462 sesiones a considerar y en por cada sesión se ejecutan unas 15 funciones. Haciendo el cálculo, para generar un cuatrimestre completo se ejecutan más de medio millón de funciones. Además, también hay que tener en cuenta que funciones como *madre* o *profesor* consumen una gran cantidad de recursos y ralentizan el proceso.

5.4 Uso de la interfaz de usuario

Para comprobar el correcto funcionamiento de la interfaz se han probado todas las opciones que ofrece.

La opción de actualización de la base de datos se probó con éxito para intervalos de días reducidos mientras que para largos intervalos se encontró el problema descrito en el apartado anterior (5.3).

En la segunda opción, consultas, se probaron todas las consultas posibles, es decir, cada una de las asignaturas, profesores, grupos y aulas. Encontrando como única anomalía la asignatura de álgebra y estadística. Cuando se consulta esta asignatura se detectan solapes inexistentes ya que en la práctica se trata como dos asignaturas separadas.

En la parte de modificaciones, se probaron con éxito todos los tipos de modificaciones en cada una de las tablas existentes.

En el apartado de comandos SQL, se probaron tanto comandos válidos como no válidos obteniendo las respuestas esperadas. Se realizaron varias modificaciones en el ámbito de la presentación de resultados de consultas ya que el programa debe estructurar los resultados en función de sus dimensiones para aprovechar el espacio de la pantalla.

6 CONCLUSIONES Y LÍNEAS FUTURAS

6.1 Valoración de la herramienta creada

La herramienta creada cumple el objetivo general de servir de apoyo a la planificación horaria de la ENM. También se han cumplido satisfactoriamente todos los objetivos específicos. La herramienta facilita tanto al profesorado como al alumnado la consulta de la sábana. Además, al almacenar la información en una base de datos, se abre un abanico de posibilidades en el cual se encuentra:

- Realizar todo tipo de consultas.
- Facilitar y hacer fiable la edición de los datos.
- Conectar con otras aplicaciones.
- Subirla a un servidor y poder acceder remotamente.

Aun con todos estos logros, se concluye que tiene limitaciones por dos motivos:

- La asignatura de álgebra y estadística: es una sola asignatura pero realmente funciona como dos asignaturas independientes haciendo que la herramienta detecte solapes inexistentes o complicando la asignación de una sesión a un profesor.
- La falta de información en el documento de planificación de las asignaturas. Cuando en una misma semana dos profesores dan clase del mismo tipo, teoría por ejemplo, es imposible saber que sesiones de teoría de la semana en cuestión le corresponden a cada uno.

6.2 Líneas futuras

En futuras versiones de la herramienta se pueden contemplar excepciones como la de álgebra y estadística pero no se va a poder arreglar el problema de la asignación de profesores a no ser que se añada información a los documentos de los que se obtienen los datos.

Entre otras ideas que han quedado pendientes se encuentran:

- Crear una interfaz gráfica más agradable para el usuario que una ventana de comandos.
- Añadir la opción de generar un fichero de icalendario con las sesiones consultadas.
- Añadir más tipos de consultas predeterminadas como la búsqueda de un profesor o grupo de clase o la búsqueda de un aula vacía.
- Reducir el tiempo de actualización de los datos.

7 BIBLIOGRAFÍA

[1] A. Moreno Vozmediano, Java para novatos: Cómo aprender programación orientada a objetos con Java sin desesperarse en el intiento, ensegundapersona.es, 2013.

[2] «TIOBE,» [En línea]. Available: https://www.tiobe.com/. [Último acceso: Febrero 2019].

[3] J. Fernández-Valdivia, «Departamento de Ciencias de la Computacion e I.A. de la Universidad de Granada,» [En línea]. Available: http://decsai.ugr.es/~jfv/ed1/c/cdrom/index.html. [Último acceso: Febrero 2019].

[4] «Códigofacilito,» [En línea]. Available: http://codigofacilito.com/. [Último acceso: Febrero 2019].

[5] «Lenguajes de Programación,» [En línea]. Available: https://lenguajesdeprogramacion.net/. [Último acceso: Febrero 2019].

[6] «pandas,» [En línea]. Available: https://pandas.pydata.org/. [Último acceso: Febrero 2019].

[7] E. Gazoni y C. Clark, «The Python Package Index (PyPI),» [En línea]. Available: https://pypi.org/project/openpyxl/. [Último acceso: Febrero 2019].

[8] B. Barragáns Martínez, «Tema 4: Introducción a los sistemas de gestión de bases de datos (SGBD),» de *Informática para la Ingeniería*, 2018.

[9] «MySQL,» [En línea]. Available: https://www.mysql.com/. [Último acceso: Febrero 2019].

[10] «DB-Egines,» Solid IT, [En línea]. Available: https://db-engines.com/en/. [Último acceso: Febrero 2019].

[11] «SQLite Tutorial,» [En línea]. Available: http://www.sqlitetutorial.net/. [Último acceso: Febrero 2019].

ANEXO I: FICHERO EXCEL.PY

```
#encoding: utf-8
from openpyxl import load_workbook, worksheet
from os import listdir
from datetime import date, timedelta, datetime


for arc in listdir():
     if '.xlsx' in arc:
          excel = arc
          break
sabana = load_workbook(excel)

for hoja in sabana.sheetnames:
     if 'SEMANA_A' in hoja:
          semanaA = hoja
          break
wsA = sabana[semanaA]

for hoja in sabana.sheetnames:
     if 'SEMANA_B' in hoja:
          semanaB = hoja
          break
wsB = sabana[semanaB]


plan = load_workbook('Estimaciones\\'+listdir('Estimaciones')[0], read_only=True)
wsp = plan[plan.sheetnames[0]]


#Función que devuelve el número de semana de curso correspondiente la fecha introducida
def semana(fecha,comienzo_cua):
     semana = fecha.isocalendar()[1] - comienzo_cua.isocalendar()[1] + 1
     if semana % 2 == 1:
          col = 'B'
     elif semana % 2 == 0:
          col = 'F'
     fil = str(((semana - 1 )// 2) * 10 + 9)
     try:
          semanap = int(wsp[col+fil].value[7:9])
     except ValueError:
          return 'Semana no lectiva.'
     return semanap


#Función que devuelve el tipo de semana (A o B) dada una semana
def tipo_semana(semana):
     col = 'B'
     fil = 9
     c = 1
     while 'I+A' not in wsp[col+str(fil)].value:
          c += 1
          col = 'F'
          if 'I+A' in wsp[col+str(fil)].value:
               break
          else:
               col = 'B'
               fil += 10
               c += 1
     imasa = c
     semanap = semana
```

```
    try:
        semanap = int(wsp[col+str(fil)].value[7:9])
    except ValueError:
        return 'Semana no lectiva.'
    if semanap < imasa:
        if semanap % 2 == 1:
            return 'A'
        else:
            return 'B'
    else:
        if semanap % 2 == 1:
            return 'B'
        else:
            return 'A'


#Función que devuelve el nombre del archivo de excel en el que se encuentra la planificación
de una asignatura
def excel_plan(asignatura):
    archivos = listdir('Estimaciones')
    for arc in archivos:
        plan = load_workbook('Estimaciones\\'+arc, read_only=True)
        wsp = plan[plan.sheetnames[0]]
        if asignatura == normalizar(wsp['B3'].value):
            return arc
    return 'No se ha encontrado el archivo de estimación.'


#Función que devuelve la posición de la planificación de una semana dada en un excel dado
def plan_semana(semana,excel):
    plan = load_workbook('Estimaciones\\'+excel, read_only=True)
    wsp = plan[plan.sheetnames[0]]
    col = 'B'
    fil = 9
    while 'SEMANA '+str(semana) not in wsp[col+str(fil)].value:
        col = 'F'
        if 'SEMANA '+str(semana) in wsp[col+str(fil)].value:
            break
        else:
            col = 'B'
            fil += 10
    return col,fil


#Función que devuelve el nombre del profesor de una clase, si es del CUD, lo hace recurriendo
al tipo de clase en el archivo del plan de la asignatura en la semana correspondienteFALLA en
TERMOD_TRANSM_CALOR)
def profesor(celda,semana,ws):
    asignatur = asignatura(celda)
    if asignatur in militares:
        fil = fila(celda)
        col = columna(celda)
        if ws.cell(fil+alto(celda,ws)-2,col).value != None:
            return ws.cell(fil+alto(celda,ws)-2,col).value
        else:
            return 'No tiene profesor asignado.'
    else:
        excel = excel_plan(asignatur)
        try:
            plan = load_workbook('Estimaciones\\'+ excel, read_only=True, data_only=True)
        except:
            return 'No se ha encontrado el archivo de estimación de esta asignatura.'
        wsp = plan[plan.sheetnames[0]]
```

```
          ref = plan_semana(semana,excel)
          if practica(celda,ws):
               if ref[0] == 'B':
                    col = 'C'
               elif ref[0] == 'F':
                    col = 'G'
          elif seminario(celda,ws):
               if ref[0] == 'B':
                    col = 'D'
               elif ref[0] == 'F':
                    col = 'H'
          elif teoria(celda,ws):
               col = ref[0]
          if wsp[col+str(ref[1]+2)].value != None:
               profesor = wsp['A'+str(ref[1]+2)].value
          elif wsp[col+str(ref[1]+3)].value != None:
               profesor = wsp['A'+str(ref[1]+3)].value
          elif wsp[col+str(ref[1]+4)].value != None:
               profesor = wsp['A'+str(ref[1]+4)].value
          elif wsp[col+str(ref[1]+5)].value != None:
               profesor = wsp['A'+str(ref[1]+5)].value
          else:
               return 'Esta hora no tiene profesor asignado esta semana.'
          return profesor


#Función que devuelve el número asociado a las letras de una columna (maximo dos letras)
def columna(celda):
     abc = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
     if len(celda.column) == 2:
          letra1 = (abc.find(celda.column[0]) + 1) * 26
          letra2 = abc.find(celda.column[1]) + 1
          return letra1 + letra2
     elif len(celda.column) == 1:
          return abc.find(celda.column[0]) + 1
     else:
          return 'Fuera de rango.'


#Función que devuelve la fila de una celda
def fila(celda):
     return celda.row


#Función que devuelve las dimensiones verticales de una celda
def alto(celda,ws):
     for rango in ws.merged_cells.ranges:
          if celda.coordinate +':' == str(rango)[0:len(celda.coordinate)+1]:
               alto = rango.size['rows']
               if alto == 2:
                    return 4
               elif alto == 5 or alto == 6:
                    return 8
               else:
                    return alto
          else:
               pass
     return 'No es una celda combinada.'


#Función que devuelve las dimensiones horizontales de una celda
def ancho(celda,ws):
     for rango in ws.merged_cells.ranges:
          if celda.coordinate +':' == str(rango)[0:len(celda.coordinate)+1]:
```

```
                return rango.size['columns']
            else:
                pass
    return 'No es una celda combinada.'


#Función que devuelve la celda que contiene el valor correspondiente a una celda combinada
def madre(celda,ws):
    if celda.value == None:
        for rango in ws.merged_cells.ranges:
            rango = rango.ref
            for fila in ws[rango][0::4]:
                if celda in fila:
                    return ws[rango.split(':')[0]]
    else:
        return celda


#Función que elimina las tildes de cadenas de caracteres en caso de tenerlas
def normalizar (palabra):
    for letra in 'ÁÉÍÓÚáéíóú':
        if letra in palabra:
            palabra = palabra.replace('á','a')
            palabra = palabra.replace('Á','A')
            palabra = palabra.replace('é','e')
            palabra = palabra.replace('É','E')
            palabra = palabra.replace('í','i')
            palabra = palabra.replace('Í','I')
            palabra = palabra.replace('ó','o')
            palabra = palabra.replace('Ó','O')
            palabra = palabra.replace('ú','u')
            palabra = palabra.replace('Ú','U')
    if ' I' in palabra[-2::1] or ' II' in palabra[-3::1]:
        palab = palabra[1:len(palabra)-2:1].lower() + palabra[-2::1]
        palabra = palabra[0] + palab[0::1]
    else:
        palabra = palabra[0] + palabra[1::1].lower()
    return palabra


#Versión de clave_de_valor() que usa una lista en vez de un diccionario
def asignatura(celda):
    nombre = celda.value
    if nombre in militares:
        return nombre
    elif nombre != None and nombre not in militares:
        p1 = nombre.split(' ')
        p1.pop()
        p2 = ''
        for letra in p1:
            p2 += letra
        nombre = p2
        for lista in asignaturas:
            if nombre in lista:
                return lista[0]
    else:
        return 'El valor no se encuentra en el diccionario'


#Función que determina si una celda consultada es de teoría
def teoria(celda,ws):
    if asignatura(celda) != 'El valor no se encuentra en el diccionario' and not
practica(celda,ws) and not seminario(celda,ws):
```

```
            return True
      else:
            return False


#Función que devuelve True si la celda consultada es una práctica(la clase de
'MECÁNICA_MÁQUINAS_FLUID' puede ser de teoría)
def practica(celda,ws):
      if asignatura(celda) != 'El valor no se encuentra en el diccionario':
            abajo = ws.cell(fila(celda)+4,columna(celda))
            arriba = ws.cell(fila(celda)-4,columna(celda))
            if celda.value == abajo.value or celda.value == arriba.value:
                  return True
            else:
                  return False
      else:
            return False


#Función que devuelve True si la celda consultada es un seminario (no funciona si solo hay un
grupo de IM en un curso con Topografía y costruccion y con Auntomoviles)
def seminario(celda,ws):
      if asignatura(celda) != 'El valor no se encuentra en el diccionario':
            col = columna(celda)
            if ancho(celda,ws) == 2:
                  return True
            elif ancho(celda,ws) == 4 and ws.cell(8, col).value == 'IM' and not
practica(celda,ws):
                  return True
            else:
                  fil = fila(celda)
                  col = 5
                  while col < ws.max_column + 1:
                        if celda.value == ws.cell(fil,col).value:
                              pass
                        elif asignatura(celda) == asignatura(ws.cell(fil,col)):
                              break
                        col += 1
                  if ancho(ws.cell(fil,col),ws) == 2:
                        return True
                  else:
                        return False
      else:
            return False


#Función que devuelve el aula en la que se imparte una clase (no funciona con: orden cerrado,
simnav, lanchas, instrucción marinera y otro tipo de clases que no se refleje el aula en la
sabana)
def aula(celda,ws):
      if celda.value == None:
            return ('Celda vacía')
      elif celda.value == 'ESTUDIO':
            return 'Biblioteca Císcar y Císcar'
      elif celda.value == 'ACTIVIDADES DE RÉGIMEN INTERIOR':
            return 'Palo/Patio Norte'
      else:
            col = columna(celda)
            fil = fila(celda)
            alt = alto(celda,ws)
            anc = ancho(celda,ws)
            if ws.cell(fil+alt-2,col+anc-1).value != None:
                  return ws.cell(fil+alt-2,col+anc-1).value
            elif ws.cell(fil+alt-2,col).value != None:
```

```
                return ws.cell(fil+alt-2,col).value
          elif ws.cell(fil+alt-3,col).value != None:
                return ws.cell(fil+alto(celda,ws)-3,col).value
          else:
                return 'No se ha encontrado el aula.'


#Función que devuelve el grupo que se encuentra verticalmente encima de la celda dada
def grupo(celda,ws):
     col = columna(celda)
     fil = 8
     if ws.cell(fil-1,col).value != None and ws.cell(fil,col).value == None:
          promocion = ws.cell(fil-1,col).value
          if promocion.count(')') != 0:
                posicion = promocion.rfind(')')+1
          else:
                posicion = len(promocion)
          return promocion[0:posicion]
     else:
          return ws.cell(fil,col).value


#Función que devuelve la brigada a la que pertenecen los alumnos que reciben una clase (no se
considera la 10ª,11ª ni 12ª brigada).
def brigada(celda,ws):
     col = columna(celda)
     fil = 7
     if ws.cell(fil,col).value == None and ws.cell(fil-1,col).value != None and
ws.cell(fila(celda),col+6).value == None:
          return 'Alumno de intercambio.'
     while ws.cell(fil,col).value == None:
          col -=1
     promocion = ws.cell(fil,col).value
     if 'AF' in promocion or 'AA' in promocion:
          return 5
     elif 'GM2' in promocion:
          return 4
     elif 'GM1' in promocion:
          return 3
     elif 'ASP2' in promocion:
          return 2
     elif 'ASP1' in promocion:
          return 1


#Función que devuelve el intervalo asignado a una clase que hay reflejado en la sabana
def hora(celda,ws):
     return ws.cell(celda.row,3).value


#Función que devuelve el día de la semana correspondiente a una clase que hay reflejado en la
sabana
def dia_semana(celda,ws):
     col = 2
     fil = fila(celda)
     while ws.cell(fil,col).value == None:
          fil -= 1
     return ws.cell(fil, col).value


#Función que convierte un número referente a un día de la semana en el nombre del día en
cuestión
def nombre_dia(numero):
```

```
    if numero == 0:
        return 'LUNES'
    elif numero == 1:
        return 'MARTES'
    elif numero == 2:
        return 'MIÉRCOLES'
    elif numero == 3:
        return 'JUEVES'
    elif numero == 4:
        return 'VIERNES'
    elif numero == 5:
        return 'SÁBADO'
    elif numero == 6:
        return 'DOMINGO'
    else:
        return 'Número fuera de rango (0,6)'


#Función que devuelve una lista con todas las celdas unitarias correspondientes a un día de la
semana
def tabla_dia(dia,ws):
    tabla = []
    tabla_cols = []
    tabla_fils = []
    fil=8
    col=5
    while col < ws.max_column + 1:
        if ws.cell(fil,col).value != None or (ws.cell(fil,col).value == None and
ancho(ws.cell(fil,col),ws) == 6):
            tabla_cols.append(ws.cell(fil,col).column)
        col += 1
    fil = 9
    col = 3
    while fil < ws.max_row + 1:
        if ws.cell(fil,col).value != None and nombre_dia(dia) ==
dia_semana(ws.cell(fil,col),ws):
            tabla_fils.append(str(ws.cell(fil,col).row))
        fil += 1
    for fil in tabla_fils:
        for col in tabla_cols:
            tabla.append(col+fil)
    return tabla

asignaturas = [
    ['Calculo I'],
    ['Fisica I'],
    ['Expresion grafica'],
    ['Calculo II'],
    ['Termodinamica y transmision de calor', 'TERMOD_TRANSM_CALOR'],
    ['Ciencia y tecnologia de materiales'],
    ['Resistencia de materiales'],
    ['Tecnologia electronica'],
    ['Ingenieria grafica'],
    ['Ingenieria de materiales'],
    ['Elasticidad y ampliacion de resistencia de materiales'],
    ['Sistemas de radiocomunicaciones', 'SIST_RADIOCOM', 'SIS_RADIO',
'SISTEMAS_RADIOCOMUNICACIONES'],
    ['Maquinas y motores navales', 'MAQ_MOT_NAVALES', 'MÁQUINAS_MOTORES_NAV.', 'MÁQ_MOT'],
    ['Topografia y construccion', 'TOPOGRAFÍA_CONSTRUCC', 'TOPO_CONST',
'TOPOGRAFÍA_CONSTRUCCIÓN'],
    ['Ingenieria de fabricacion y calidad dimensional', 'ING_FAB_CAL_DIM',
'ING_FABRIC_CAL_DIMENS'],
    ['Teoria de estructuras y construcciones industriales'],
    ['Oficina tecnica'],
```

```
     ['Sistemas de control y sensores navales'],
     ['Ampliacion de informatica'],
     ['Instalaciones y construccion naval'],
     ['Automoviles'],
     ['Fisica II', 'FÍSICAII(AULA40)/ÁLGEBRAYESTADÍSTICA(AULA', 'FÍSICAII'],
     ['Algebra y estadistica', 'ESTADÍSTICA', 'ÁLGEBRA',
'FÍSICAII(AULA40)/ÁLGEBRAYESTADÍSTICA(AULA'],
     ['Quimica', 'QUÍMICA'],
     ['Informatica para la ingenieria', 'INFORMÁTICA_INGENIERÍA', 'INFORMÁTICA'],
     ['Introduccion a la gestion empresarial', 'INTRODUCCIÓN_GESTIÓN_EMPRESARIAL',
'INTROD_GESTIÓN_EMPRESARIAL', 'I_GEST_EMPRES'],
     ['Mecanica de fluidos', 'MEC_FLUIDOS', 'M_FLUIDOS', 'MECÁNICA_FLUIDOS'],
     ['Ingles I', 'INGLÉSIgruposdenivel1,2,3y4'],
     ['Teoria de maquinas y mecanismos', 'T_MÁQUINAS_MECANISMOS', 'T_MÁQUINAS_MEC', 'T_MÁQ'],
     ['Fundamentos de electrotecnia', 'FUNDAM_ELECTROTECNIA', 'F_ELECTROTECNIA', 'ELECTROT.'],
     ['Tecnologia medioambiental', 'T_MEDIOAMBIENTAL', 'T_MEDIOAMB'],
     ['Fundamentos de automatica'],
     ['Fundamentos de sistemas y tecnologias de fabricacion'],
     ['Ingenieria termica I'],
     ['Maquinas de fluidos'],
     ['Ingles II', 'INGLÉSIIgruposdenivel1,2,3y4'],
     ['Fundamentos de organizacion de empresas'],
     ['Diseño de maquinas', 'DISEÑODEMÁQUINAS', 'DISEÑO_MÁQ.'],
     ['Mecánica y máquinas de fluidos', 'MECÁNICA_MÁQUINAS_FLUID']
     ]

militares = ['ESTUDIO', 'TFG', 'ACTIVIDADES DE RÉGIMEN INTERIOR', 'TÁCTICA NAVAL III G1 G2 G3
G4 G5 G6', 'OPERACIONES ANFIBIAS II', 'TÁCTICA NAVAL III (AF)', 'CONTABILIDAD_ARMADA',
'MANIOBRA_NAVEGACIÓN_I', 'MANIOBRA_NAVEGACIÓN_IM II', 'GESTIÓN_REC_PERSONAL', 'MANIOBRA_NAVEG
II (MILCOM)', 'MANIOBRA_NAVEG II', 'TÁCTICA ANFIBIA IV', 'MANIOBRA Y NAVEGACIÓN II',
'CONTRATACIÓN_PÚBLICA', 'OPERACIONES_ANFIBIAS_I', 'GESTIÓN_PRESUPUESTARIA',
'LENGUA_ESPAÑOLA_II', 'TÁCTICA_NAVAL', 'GEST_ESCALONES_APROV', 'HISTORIA NAVAL', 'INGLÉS',
'CONTROL_MAT_APROV', 'LOGÍSTICA_GENERAL', 'COMUNICACIONES', 'INGLÉS III', 'INGLÉS II',
'FORMACIÓN_MILITAR_I', 'MANIOBRA Y NAVEGACIÓN', 'SISTEMAS DE ARMAS Y TN', 'CONTRAT_PÚBLICA',
'MANIOBRA_NAVEG_I', 'LENGUA INGLESA V', 'FORMACIÓN MILITAR III G1G2G3', 'SEGURIDAD INTERIOR I
G4G5G6', 'LIDERAZGO', 'TÁCTICA_ANFIBIA_I (ASP2)', 'TÁCTICA ANFIBIA II', 'RETRIB_INDEMNIZ',
'LENGUA_ESPAÑOLA_I', 'SEGURIDAD INTERIOR I G1G2G3', 'SISTEMAS DE ARMAS Y TN I G4G5G6',
'SEGURIDAD INTERIOR I (GM2)', 'MANIOBRA_NAVEGACIÓN_II (ASP2)', 'MAN_NAVEG_II G1G2', 'SEGURIDAD
INTERIOR', 'SISTEMAS ARMAS IM', 'SISTEMAS_ARMAS IM (GM2)', 'GESTIÓN_ECONÓMICA_SEA',
'COMUNICACIONES IM II', 'SIMTAC', 'COMUNICACIONES_IM II (AA)', 'MAN_NAVEG_II G3G4',
'MAN_NAVEG_I G1G2G3G4', 'GEST_ESCAL_APROV', 'TÁCTICA ANFIBIA III', 'TÁCTICA_ANFIBIA_I (GM2)',
'MAN_NAVEG_I G5G6', 'TÁCTICA ANFIBIA I', 'CONTR_MAT_APROV', 'MAN_NAVEG_II G5G6',
'CONTAB_ARMADA', 'LOGÍSTICA_GEST_REC', 'TOPOGRAFÍA (ASP2)', 'TOPOGRAFIA_II',
'FORMACIÓN_MILITAR_II', 'ARTILLERÍA Y COORD_FUEGOS', 'DERECHO_MARÍTIMO',
'FORMACIÓN_MILITAR_III', 'FORMACIÓN MILITAR III G4G5G6', 'SISTEMAS DE ARMAS Y TN I G1G2G3',
'SISTEMAS DE ARMAS Y TN I (GM2)', 'SEGOP/PRL', 'ZAPADORES (ASP2)', 'ZAPADORES',
'PLATAFORMA_NAVAL', 'FORMACIÓN MILITAR  III G4G5G6']

lista_aulas = [[1, 0, 45], [2, 0, 44], [3, 1, 20], [4, 1, 24], [5, 0, 63], [6, 0, 15], [7, 0,
30], [8, 0, 30], [9, 0, 90], [10, 0, 44], [11, 0, 0], [12, 0, 42], [13, 1, 20], [14, 1, 24],
[15, 1, 24], [16, 0, 45], [17, 1, 24], [18, 1, 24], [19, 0, 26], [20, 0, 0], [30, 0, 45], [40,
0, 48], [41, 0, 27], [42, 0, 44], [43, 1, 24], [44, 1, 24], [45, 0, 62], [50, 1, 48], [51, 0,
54], [52, 0, 58], [53, 0, 45], [54, 0, 61], ['55/57', 0, 84], [56, 0, 62], ['Lab.
Electricidad', 1, 20], ['S1 IP', 0, 10], ['S2 IP', 0, 9], ['S3 IP', 0, 9], ['S4 IP', 0, 8],
['Lab. Quimica', 0, 26], ['Lab. Fisica', 1, 20], ['Aparatos Auxiliares', 0, 18], ['Turbinas',
0, 18], ['Motores', 0, 22], ['Lab. Materiales', 0, 20], ['S1', 0, 16], ['S2', 0, 24], ['S3',
0, 24], ['S4', 0, 23], ['BA', 0, 20]]


def tabla_asignaturas():
     tabla = []
     c = 0
     for asig in asignaturas:
```

```
            tabla.append([c+1])
            tabla[c].append(asig[0])
            c += 1
      return tabla


def tabla_profs():
      from urllib.request import urlopen
      from html.parser import HTMLParser

      try:
            # Abrir URL.
            r =
urlopen("https://cud.uvigo.es/index.php?option=com_content&view=article&id=52&Itemid=53")
            # Leer el contenido y convertirlo en str
            pagina = str(r.read())
      except:
            tabla = [[1, 'Alfonsín Pérez', 'Víctor Ángel', '986804942', '8244942',
'valfonsin@cud.uvigo.es'], [2, 'Álvarez Feijoo', 'Miguel Ángel', '986804920', '8244920',
'alvarezfeijoo@cud.uvigo.es'], [3, 'Álvarez Hernández', 'María', '986804923', '8244923',
'maria.alvarez@cud.uvigo.es'], [4, 'Arce Fariña', 'Elena', '986804939', '8244939',
'elena.arce@cud.uvigo.es'], [5, 'Barragáns Martínez', 'Belén', '986804903', '8244903',
'belen@cud.uvigo.es'], [6, 'Bellas Rivera', 'Roberto', '986804945', '8244945',
'rbellas@cud.uvigo.es'], [7, 'Cacabelos Reyes', 'Antón', '986804946', '8244946',
'acacabelos@cud.uvigo.es'], [8, 'Carreño Morales', 'Rafael María', '986804924', '8244924',
'rafaelm@cud.uvigo.es'], [9, 'Casqueiro Placer', 'Carlos', '986804935', '8244935',
'ccasqueiro@cud.uvigo.es'], [10, 'Cocheteux Lourido', 'Roberto Ramón', '986804948', '8244948',
'rcoclou@cud.uvigo.es'], [11, 'Devesa Rey', 'Rosa', '986804936', '8244936',
'rosa.devesa.rey@cud.uvigo.es'], [12, 'Eirís Barca', 'Antonio', '986804926', '8244926',
'eiris@cud.uvigo.es'], [13, 'Febrero Garrido', 'Lara', '986804918', '8244918',
'lfebrero@cud.uvigo.es'], [14, 'Fernández García', 'Norberto', '986804922', '8244922',
'norberto@cud.uvigo.es'], [15, 'Fernández Gavilanes', 'Milagros', '986804921', '8244921',
'mfgavilanes@cud.uvigo.es'], [16, 'Gómez Pérez', 'Paula', '986804938', '8244938',
'paula@cud.uvigo.es'], [17, 'Gómez Rodríguez', 'Miguel Ángel', '986804943', '8244943',
'miguelgr@cud.uvigo.es'], [18, 'González Gil', 'Arturo', '986804947', '8244947',
'arturogg@cud.uvigo.es'], [19, 'González Valdivia', 'Fernando', '986804902', '8244902',
'gerente@cud.uvigo.es'], [20, 'Guzmán Crespo', 'Francisco Javier', '986804948', '8244948',
'fguzcre@cud.uvigo.es'], [21, 'Lareo Calviño', 'Guillermo', '986804917', '8244917',
'glareo@cud.uvigo.es'], [22, 'Maceiras Castro', 'Rocío', '986804933', '8244933',
'rmaceiras@cud.uvigo.es'], [23, 'Núñez Nieto', 'Xavier', '986804925', '8244925',
'xnnieto@cud.uvigo.es'], [24, 'Núñez Ortuño', 'José María', '986804937', '8244937',
'jnunez@cud.uvigo.es'], [25, 'Pousada Carballo', 'José María', '986804901', '8244901',
'chema@cud.uvigo.es'], [26, 'Puente Luna', 'Iván', '986804848', '8244848',
'ipuente@cud.uvigo.es'], [27, 'Rey González', 'Guillermo David', '986804941', '8244941',
'guillermo.rey@cud.uvigo.es'], [28, 'Rodelgo Lacruz', 'Miguel', '986804949', '8244949',
'mrodelgo@cud.uvigo.es'], [29, 'Rodríguez Rodríguez', 'Francisco Javier', '986804916',
'8244916', 'fjavierrodriguez@cud.uvigo.es'], [30, 'Solla Carracelas', 'María Mercedes',
'986804940', '8244940', 'merchisolla@cud.uvigo.es'], [31, 'Suárez García', 'Andrés',
'986804934', '8244934', 'asuarez@cud.uvigo.es'], [32, 'Touza Gil', 'Ramón', '986804950',
'8244950', 'rtougil@cud.uvigo.es'], [33, 'Urréjola Madriñán', 'Santiago', '986804904',
'8244904', 'urrejola@cud.uvigo.es'], [34, 'Vázquez Carpentier', 'Alicia', '986804923',
'8244923', 'avcarpentier@cud.uvigo.es'], [35, 'Beasley', 'Jeffrey G.', '986804927', '8244927',
'externo.jgbeasley@cud.uvigo.es'], [36, 'Foley', 'Mary', '986804927', '8244927',
'externo.mfoley@cud.uvigo.es'], [37, 'Rich Stephens', 'Martyn', '986804927', '8244927',
'externo.martynrich@cud.uvigo.es'], [38, 'Tomé Rosales', 'Ángeles', '986804927', '8244927',
'externo.angelestome@cud.uvigo.es']]
            print('Fallo en conexión a internet, tabla de profesores creada con valores
predeterminados.')
            return tabla
      #Crear listas
      datos = []
      lista = []
      #Definir el parser para que trate solo los datos explicitos de la pagina
      class MyHTMLParser(HTMLParser):
```

```
        def handle_data(self, data):
            datos.append(data)
    #Introducir pagina en parser
    MyHTMLParser().feed(pagina)

    # Cerrar para liberar recursos.
    r.close()

    #Función que elimina las tildes de cadenas de caracteres en caso de tenerlas
    def caracteres_especiales (palabra):
        for expresion in
['\\xc3\\xa1','\\xc3\\x81','\\xc3\\xa9','\\xc3\\x89','\\xc3\\xad','\\xc3\\x8d','\\xc3\\xb3','\
\xc3\\x93','\\xc3\\xba','\\xc3\\x9a','\\xc3\\xb1']:
            if expresion in palabra:
              palabra = palabra.replace('\\xc3\\xa1','á')
              palabra = palabra.replace('\\xc3\\x81','Á')
              palabra = palabra.replace('\\xc3\\xa9','é')
              palabra = palabra.replace('\\xc3\\x89','É')
              palabra = palabra.replace('\\xc3\\xad','í')
              palabra = palabra.replace('\\xc3\\x8d','Í')
              palabra = palabra.replace('\\xc3\\xb3','ó')
              palabra = palabra.replace('\\xc3\\x93','Ó')
              palabra = palabra.replace('\\xc3\\xba','ú')
              palabra = palabra.replace('\\xc3\\x9a','Ú')
              palabra = palabra.replace('\\xc3\\xb1','ñ')
        return palabra

    #Filtrar y corregir los datos obtenidos a traves del parser
    c = 0
    for elem in datos:
         if elem == '\\xc2\\xa0' or 'Conserjer' in elem:
              c = 2
         elif c == 1 and '\\r\\n' not in elem:
              elem = caracteres_especiales(elem)
              if elem[0] == ' ':
                   elem = elem.replace(' ', '')
              lista.append(elem)
         elif elem == 'Email':
              c = 1
         elif elem == '\\xc2\\xa0':
              c = 2

    #Convertir datos obtenidos en formato tabla(tambien se corrige el espacio en los números
de teléfono)
    tabla = []
    c = 0
    for elem in lista:
         if lista.index(elem)%5 == 0:
              tabla.append([c+1])
              tabla[c].append(elem)
         if lista.index(elem)%5 == 1:
              tabla[c].append(elem)
         if lista.index(elem)%5 == 2:
              telefono = elem.replace(' ', '')
              tabla[c].append(telefono)
         if lista.index(elem)%5 == 3:
              tabla[c].append(elem)
         if lista.index(elem)%5 == 4:
              tabla[c].append(elem)
              c += 1
    return tabla
```

```
def tabla_grupos(ws=wsA):
     tabla = []
     c = 0
     fil=8
     col=5
     while col < ws.max_column + 1:
          if ws.cell(fil,col).value != None or (ws.cell(fil,col).value == None and
ancho(ws.cell(fil,col),ws) == 6):
               grupot = grupo(ws.cell(fil,col),ws)
               curso = brigada(ws.cell(fil,col),ws)
               tabla.append([c+1])
               tabla[c].append(curso)
               tabla[c].append(grupot)
               tabla[c].append(0)
               c += 1
          col += 1
     return tabla


def tabla_aulas():
     c = 0
     for tup in lista_aulas:
          tup.insert(0, c+1)
          c += 1
     return lista_aulas


def tabla_sesiones(comienzo_cua,comienzo_act,fin_act):
     tabla =[]
     c = 0
     fecha = comienzo_act
     while fecha <= fin_act:
          print(str(fecha))
          sem = semana(fecha,comienzo_cua)
          if fecha.weekday() != 5 and fecha.weekday() != 6 and type(sem) == int:
               tipo = tipo_semana(sem)
               if tipo == 'A':
                    ws = wsA
               elif tipo == 'B':
                    ws = wsB
               else:
                    continue
               for sesion in tabla_dia(fecha.weekday(),ws):
                    celd = madre(ws[sesion],ws)
                    if celd.value == None:
                         print(' No se ha encontrado el nombre de la asignatura de la sesion
correspondiente a la celda: '+celd.coordinate)
                         continue
                    tabla.append([c+1])
                    tabla[c].append(str(fecha))
                    tabla[c].append(hora(ws[sesion],ws))
                    tabla[c].append(asignatura(celd))
                    tabla[c].append(profesor(celd,sem,ws))
                    tabla[c].append(brigada(celd,ws))
                    tabla[c].append(grupo(ws[sesion],ws))
                    tabla[c].append(aula(celd,ws))
                    c += 1
          fecha += timedelta(days=1)
        return tabla
```

ANEXO II: FICHERO SQLITE.PY

```
#encoding: utf-8
import sqlite3
from datetime import datetime, timedelta, date
from excel import tabla_grupos, tabla_asignaturas,tabla_aulas, tabla_sesiones, tabla_profs
from os import system

while True:
	system('cls')
	print('\n Introduzca la fecha donde comienza el cuatrimestre: ')
	comienzo_cuad = int(input(' Día: '))
	if comienzo_cuad == 0:
		continue
	comienzo_cuam = int(input(' Mes: '))
	if comienzo_cuam == 0:
		continue
	comienzo_cuaa = int(input(' Año: '))
	if comienzo_cuaa == 0:
		continue
	print('\n Introduzca la fecha donde comienza la actualización: ')
	comienzo_d = int(input(' Día: '))
	if comienzo_d == 0:
		continue
	comienzo_m = int(input(' Mes: '))
	if comienzo_m == 0:
		continue
	comienzo_a = int(input(' Año: '))
	if comienzo_a == 0:
		continue
	print('\n Introduzca la fecha donde finaliza la actualización: ')
	fin_d = int(input(' Día: '))
	if fin_d == 0:
		continue
	fin_m = int(input(' Mes: '))
	if fin_m == 0:
		continue
	fin_a = int(input(' Año: '))
	if fin_a == 0:
		continue
	else:
		try:
			comienzo_cua = date(comienzo_cuaa,comienzo_cuam,comienzo_cuad)
			comienzo_act = date(comienzo_a,comienzo_m,comienzo_d)
			fin_act = date(fin_a,fin_m,fin_d)
		except:
			print('\n ERROR EN FECHAS')
			input()
			continue
		break


print('\n Esto puede tardar un rato...')
t0 = datetime.now()


#Se ejecuta la generación de los datos para las tablas
print(' Generando datos...')
print(' Profesores...')
tablaP = tabla_profs()
print(' Asignaturas...')
```

```
tablaA = tabla_asignaturas()
print(' Grupos...')
tablaG = tabla_grupos()
print(' Aulas...')
tablaAu = tabla_aulas()
print(' Sesiones...')
tablaS = tabla_sesiones(comienzo_cua,comienzo_act,fin_act)
print(' Datos generados con éxito.')

print(' Filtrando datos...')
for tup in tablaS:
	for asig in tablaA:
		if tup[3] in asig:
			tup[3] = asig[0]
	if tup[4] == 'No tiene profesor asignado.' or tup[4] == 'No se ha encontrado el archivo
de estimación de esta asignatura.':
		tup[4] = 'Desconocido'
	else:
		for prof in tablaP:
			if str(prof[1]) in str(tup[4]):
				tup[4] = prof[0]
	for grup in tablaG:
		if tup[5] == grup[1] and tup[6] == grup[2]:
			tup[5] = grup[0]
			tup.pop(6)
	for aul in tablaAu:
		if tup[6] == aul[1]:
			tup[6] = aul[0]
print(' Datos filtrados con éxito.')

#Creación de las tablas
print(' Creando tablas...')
conexion = sqlite3.connect("sabana.db")
cursor = conexion.cursor()
cursor.execute("DROP TABLE IF EXISTS profesores")
cursor.execute("CREATE TABLE IF NOT EXISTS profesores (idprofesores INT(3) NOT NULL, apellidos
VARCHAR(100) NOT NULL, nombre VARCHAR(100) NOT NULL, telefono INT(9) NULL, rpv INT(8) NULL,
email VARCHAR(60) NULL, PRIMARY KEY (idprofesores))")

cursor.execute("DROP TABLE IF EXISTS asignaturas")
cursor.execute("CREATE TABLE IF NOT EXISTS asignaturas (idasignaturas INT(3) NOT NULL, nombre
VARCHAR(100) NOT NULL, PRIMARY KEY (idasignaturas))")

cursor.execute("DROP TABLE IF EXISTS grupos")
cursor.execute("CREATE TABLE IF NOT EXISTS grupos (idgrupos INT(3) NOT NULL, curso INT(2) NOT
NULL, grupo VARCHAR(40) NOT NULL, cantidad_alumnos INT(2) NULL, PRIMARY KEY (idgrupos))")

cursor.execute("DROP TABLE IF EXISTS aulas")
cursor.execute("CREATE TABLE IF NOT EXISTS aulas (idaulas INT(3) NOT NULL, nombre VARCHAR(40)
NOT NULL, ordenadores BOOLEAN NOT NULL, sillas INT(2) NOT NULL, PRIMARY KEY (idaulas))")

cursor.execute("DROP TABLE IF EXISTS sesiones")
cursor.execute("CREATE TABLE IF NOT EXISTS sesiones (idsesiones INT(7) NOT NULL, fecha
VARCHAR(11) NOT NULL, hora VARCHAR(10) NOT NULL, asignatura VARCHAR(80) NULL, profesor
VARCHAR(80) NULL, grupo VARCHAR(30) NOT NULL, aula VARCHAR(30) NULL, PRIMARY KEY
(idsesiones))")
print(' Tablas creadas con éxito.')



#Se rellenan las tablas
print(' Rellenando tablas...')
for fila in tablaP:
```

```
    orden = "INSERT INTO profesores(idprofesores,apellidos,nombre,telefono,rpv,email)
VALUES("+str(fila[0])+",'"+fila[1]+"','"+fila[2]+"',"+fila[3]+","+fila[4]+",'"+fila[5]+"')"
    cursor.execute(orden)

for fila in tablaA:
    orden = "INSERT INTO asignaturas(idasignaturas,nombre)
VALUES("+str(fila[0])+",'"+fila[1]+"')"
    cursor.execute(orden)

for fila in tablaG:
       if type(fila[1]) == str:
           fila[1] = 0
       orden = "INSERT INTO grupos(idgrupos,curso,grupo,cantidad_alumnos)
VALUES("+str(fila[0])+","+str(fila[1])+",'"+fila[2]+"',"+str(fila[3])+")"
       cursor.execute(orden)

for fila in tablaAu:
    orden = "INSERT INTO aulas(idaulas,nombre,ordenadores,sillas)
VALUES("+str(fila[0])+",'"+str(fila[1])+"',"+str(fila[2])+","+str(fila[3])+")"
    cursor.execute(orden)

for fila in tablaS:
    orden = "INSERT INTO sesiones(idsesiones,fecha,hora,asignatura,profesor,grupo,aula)
VALUES("+str(fila[0])+",'"+str(fila[1])+"','"+fila[2]+"','"+str(fila[3])+"','"+str(fila[4])+"'
,'"+str(fila[5])+"','"+str(fila[6])+"')"
    cursor.execute(orden)
print(' Tablas rellenas con éxito.')


conexion.commit()



conexion.close()
tf = datetime.now() - t0
print(' Base de datos actulizada con éxito.\n Tiempo transcurrido:',tf)
```

ANEXO III: FICHERO CONSULTAS.PY

```
#encoding: utf-8
from os import system

system('cls')
print('\n+---------------------------------------------+\n| BIENVENIDO A LA CONSULTA DE SÁBANA
v.0.14.9 |\n+---------------------------------------------+')
while True:
	print('\n 1 - Actualizar base de datos\n 2 - Hacer una consulta\n 3 - Modificar base de
datos\n 4 - SQL\n 0 - Salir')
	eleccion0 = input('\n ¿Qué desea hacer? ')

	if eleccion0 == '1':
		system('cls')
		import sqlite
		input()



	elif eleccion0 == '2':
		while True:
			import sqlite3
			conexion = sqlite3.connect("sabana.db")
			cursor = conexion.cursor()
			system('cls')
			print('\n Indique los días que quiere consultar: \n')
			inicio = input(' Fecha de inicio (AAAA-MM-DD): ')
			if inicio == '0':
				break
			fin = input('\n Fecha de fin (AAAA-MM-DD): ')
			if fin == '0':
				break
			while True:
				system('cls')
				print('\n CONSULTA DE LA BASE DE DATOS\n\n 1 - Asignatura\n 2 -
Profesor\n 3 - Grupo\n 4 - Aula\n 0 - Atrás')
				eleccion2 = input('\n Elija una opción: ')
				system('cls')


				if eleccion2 == '1':
					cursor.execute("SELECT * FROM asignaturas ORDER BY nombre")
					for elem in cursor.fetchall():
						print(' {:3} -> {}'.format(elem[0],elem[1]))
					eleccion2_1 = input('\n Elija asignatura: ')
					if eleccion2_1 == '0':
						break
					system('cls')
					cursor.execute("SELECT fecha,hora,CASE
typeof(profesores.nombre) WHEN typeof(null) THEN '...' ELSE profesores.nombre END,CASE
typeof(profesores.apellidos) WHEN typeof(null) THEN 'Desconocido' ELSE profesores.apellidos
END,grupos.curso,grupos.grupo,CASE typeof(aulas.nombre) WHEN typeof(null) THEN aula ELSE
aulas.nombre END AS aula FROM sesiones LEFT OUTER JOIN profesores ON sesiones.profesor =
profesores.idprofesores LEFT OUTER JOIN grupos ON sesiones.grupo = grupos.idgrupos LEFT OUTER
JOIN aulas ON sesiones.aula = aulas.idaulas WHERE sesiones.asignatura=" + eleccion2_1+" AND
fecha BETWEEN '"+inicio+"' AND '"+fin+"'")
					resultado = cursor.fetchall()
					longitud = len(resultado)
					for elem in resultado:
						indice = resultado.index(elem)
```

```
                                        for ref in range(indice,longitud):
                                                if elem[0] in resultado[ref] and elem[1] in
resultado[ref] and indice != ref and elem[6] in resultado[ref] and elem[3] in resultado[ref]:
                                                        elem = list(elem)
                                                        elem[5] = elem[5]+','+resultado[ref][5]
                                                        resultado[indice] = tuple(elem)
                                                        resultado[ref] = [str(ref),
'ERROR','CLASS', 'ERROR','CLASS', 'ERROR','CLASS', 'ERROR','CLASS', 'ERROR']
                                        for eleme in resultado:
                                                if type(eleme) == list:
                                                        resultado.remove(eleme)
                                        titulos = []
                                        for col in cursor.description:
                                                titulos.append(col[0].upper())
                                        print('\n {:10}  {:9}    {:37}  {:2}  {:15}
{:20}'.format(str(titulos[0]),str(titulos[1]),'PROFESOR',str(titulos[4]),str(titulos[5]),str(t
itulos[6])),'\n')
                                        for elem in resultado:
                                                if len(elem) == len(resultado[0]):
                                                        print(' {:10}  {:9}    {:40}  {:2}  {:15}
{:20}'.format(str(elem[0]),str(elem[1]),str(elem[3])+',
'+str(elem[2]),str(elem[4]),str(elem[5]),str(elem[6])))
                                        longitud = len(resultado)
                                        for elem in resultado:
                                                indice = resultado.index(elem)
                                                for ref in range(indice,longitud):
                                                        if elem[0] in resultado[ref] and elem[1] in
resultado[ref] and indice != ref and ((elem[6] in resultado[ref] and elem[3] not in
resultado[ref]) or (elem[6] not in resultado[ref] and elem[3] in resultado[ref])):
                                                                print('\n La asignatura seleccionada
aparece en dos aulas o profesores diferentes para el mismo intervalo de tiempo: \n')
                                                                print(' {:10}  {:9}    {:40}  {:2}
{:15}  {:20}'.format(str(elem[0]),str(elem[1]),str(elem[3])+',
'+str(elem[2]),str(elem[4]),str(elem[5]),str(elem[6])))
                                                                print(' {:10}  {:9}    {:40}  {:2}
{:15}  {:20}'.format(str(resultado[ref][0]),str(resultado[ref][1]),str(resultado[ref][3])+',
'+str(resultado[ref][2]),str(resultado[ref][4]),str(resultado[ref][5]),str(resultado[ref][6]))
)
                                        input()


                                elif eleccion2 == '2':
                                        print('\n {}  {}'.format('NÚMERO',' PROFESOR\n'))
                                        cursor.execute("SELECT * FROM profesores")
                                        for elem in cursor.fetchall():
                                                print(' {:3} -> {},
{}'.format(elem[0],elem[1],elem[2]))
                                        eleccion2_2 = input('\n Elija profesor: ')
                                        if eleccion2_2 == '0':
                                                break
                                        system('cls')
                                        cursor.execute("SELECT fecha,hora,asignaturas.nombre AS
asignatura,grupos.curso,grupos.grupo,CASE typeof(aulas.nombre) WHEN typeof(null) THEN aula
ELSE aulas.nombre END AS aula FROM sesiones LEFT OUTER JOIN asignaturas ON sesiones.asignatura
= asignaturas.idasignaturas LEFT OUTER JOIN grupos ON sesiones.grupo = grupos.idgrupos LEFT
OUTER JOIN aulas ON sesiones.aula = aulas.idaulas WHERE sesiones.profesor=" + eleccion2_2+"
AND fecha BETWEEN '"+inicio+"' AND '"+fin+"'")
                                        resultado = cursor.fetchall()
                                        longitud = len(resultado)
                                        for elem in resultado:
                                                indice = resultado.index(elem)
                                                for ref in range(indice,longitud):
```

```
                                                if elem[0] in resultado[ref] and elem[1] in
resultado[ref] and indice != ref and elem[5] in resultado[ref] and elem[2] in resultado[ref]:
                                                        elem = list(elem)
                                                        elem[4] = elem[4]+','+resultado[ref][4]
                                                        resultado[ref] = [str(ref),
'ERROR','CLASS', 'ERROR','CLASS', 'ERROR','CLASS', 'ERROR','CLASS', 'ERROR']
                                                        resultado[indice] = tuple(elem)
                                        for eleme in resultado:
                                                if type(eleme) == list:
                                                        resultado.remove(eleme)
                                        titulos = []
                                        for col in cursor.description:
                                                titulos.append(col[0].upper())
                                        print('\n {:10}  {:9}  {:37}  {:2}  {:15}
{:20}'.format(*titulos),'\n')
                                        for elem in resultado:
                                                if len(elem) == len(resultado[0]):
                                                        print(' {:10}  {:9}  {:40}  {:2}  {:15}
{:20}'.format(str(elem[0]),str(elem[1]),str(elem[2]),str(elem[3]),str(elem[4]),str(elem[5])))
                                        longitud = len(resultado)
                                        for elem in resultado:
                                                indice = resultado.index(elem)
                                                for ref in range(indice,longitud):
                                                        if elem[0] in resultado[ref] and elem[1] in
resultado[ref] and indice != ref and (elem[5] not in resultado[ref] or elem[2] not in
resultado[ref]):
                                                                print('\n El profesor seleccionado se
aparece en dos aulas o asignaturas diferentes en el mismo intervalo de tiempo: \n')
                                                                print(' {:10}  {:9}  {:40}  {:2}  {:15}
{:20}'.format(str(elem[0]),str(elem[1]),str(elem[2]),str(elem[3]),str(elem[4]),str(elem[5])))
                                                                print(' {:10}  {:9}  {:40}  {:2}  {:15}
{:20}'.format(str(resultado[ref][0]),str(resultado[ref][1]),str(resultado[ref][2]),str(resulta
do[ref][3]),str(resultado[ref][4]),str(resultado[ref][5])))
                                        input()


                                elif eleccion2 == '3':
                                        cursor.execute("SELECT * FROM grupos")
                                        for elem in cursor.fetchall():
                                                print(' {:3} -> {}
{}'.format(elem[0],elem[1],elem[2]))
                                        eleccion2_3 = input('\n Elija grupo: ')
                                        if eleccion2_3 == '0':
                                                break
                                        system('cls')
                                        cursor.execute("SELECT fecha,hora,CASE
typeof(asignaturas.nombre) WHEN typeof(null) THEN asignatura ELSE asignaturas.nombre END AS
asignatura,CASE typeof(profesores.nombre) WHEN typeof(null) THEN ' ' ELSE profesores.nombre
END,CASE typeof(profesores.apellidos) WHEN typeof(null) THEN profesor ELSE
profesores.apellidos END,CASE typeof(aulas.nombre) WHEN typeof(null) THEN aula ELSE
aulas.nombre END AS aula FROM sesiones LEFT OUTER JOIN asignaturas ON sesiones.asignatura =
asignaturas.idasignaturas LEFT OUTER JOIN profesores ON sesiones.profesor =
profesores.idprofesores LEFT OUTER JOIN aulas ON sesiones.aula = aulas.idaulas WHERE
sesiones.grupo=" + eleccion2_3+" AND fecha BETWEEN '"+inicio+"' AND '"+fin+"'")
                                        resultado = cursor.fetchall()
                                        titulos = []
                                        for col in cursor.description:
                                                titulos.append(col[0].upper())
                                        print('\n {:10}  {:9}  {:40}  {:40}
{:20}'.format(str(titulos[0]),str(titulos[1]),str(titulos[2]),'PROFESOR',str(titulos[5])),'\n'
)
                                        for elem in resultado:
```

```
                                        print(' {:10}  {:9}  {:40}  {:40}
{:20}'.format(str(elem[0]),str(elem[1]),str(elem[2]),str(elem[4])+',
'+str(elem[3]),str(elem[5])))
                                longitud = len(resultado)
                                for elem in resultado:
                                        indice = resultado.index(elem)
                                        for ref in range(indice,longitud):
                                                if elem[0] in resultado[ref] and elem[1] in
resultado[ref] and indice != ref:
                                                        print('El grupo seleccionado se
encuentra tiene dos sesiones diferentes en el mismo intervalo de tiempo:\n')
                                                        print(' {:10}  {:9}  {:40}  {:40}
{:20}'.format(str(elem[0]),str(elem[1]),str(elem[2]),str(elem[4])+',
'+str(elem[3]),str(elem[5])))
                                                        print(' {:10}  {:9}  {:40}  {:40}
{:20}'.format(str(resultado[ref][0]),str(resultado[ref][1]),str(resultado[ref][2]),str(resulta
do[ref][4])+', '+str(resultado[ref][3]),str(resultado[ref][5])))
                                input()


                        elif eleccion2 == '4':
                                cursor.execute("SELECT * FROM aulas")
                                for elem in cursor.fetchall():
                                        print(' {:3} -> {}'.format(elem[0],elem[1]))
                                eleccion2_4 = input('\n Elija aula: ')
                                if eleccion2_4 == '0':
                                        break
                                system('cls')
                                cursor.execute("SELECT fecha,hora,CASE
typeof(asignaturas.nombre) WHEN typeof(null) THEN asignatura ELSE asignaturas.nombre END AS
asignatura,CASE typeof(profesores.nombre) WHEN typeof(null) THEN ' ' ELSE profesores.nombre
END,CASE typeof(profesores.apellidos) WHEN typeof(null) THEN profesor ELSE
profesores.apellidos END,grupos.curso,grupos.grupo FROM sesiones LEFT OUTER JOIN asignaturas
ON sesiones.asignatura = asignaturas.idasignaturas LEFT OUTER JOIN profesores ON
sesiones.profesor = profesores.idprofesores LEFT OUTER JOIN grupos ON sesiones.grupo =
grupos.idgrupos WHERE sesiones.aula=" + eleccion2_4+" AND fecha BETWEEN '"+inicio+"' AND
'"+fin+"'")
                                resultado = cursor.fetchall()
                                longitud = len(resultado)
                                for elem in resultado:
                                        indice = resultado.index(elem)
                                        for ref in range(indice,longitud):
                                                if elem[0] in resultado[ref] and elem[1] in
resultado[ref] and indice != ref and elem[4] in resultado[ref]:
                                                        elem = list(elem)
                                                        elem[6] = elem[6]+','+resultado[ref][6]
                                                        resultado[ref] = [str(ref),
'ERROR','CLASS', 'ERROR','CLASS', 'ERROR','CLASS', 'ERROR','CLASS', 'ERROR']
                                                        resultado[indice] = tuple(elem)
                                for eleme in resultado:
                                        if type(eleme) == list:
                                                resultado.remove(eleme)
                                titulos = []
                                for col in cursor.description:
                                        titulos.append(col[0].upper())
                                print('\n {:10}  {:9}  {:40}  {:37}  {:2}
{:15}'.format(str(titulos[0]),str(titulos[1]),str(titulos[2]),'PROFESOR',str(titulos[5]),str(t
itulos[6])),'\n')
                                for elem in resultado:
                                        if len(elem) == len(resultado[0]):
                                                print(' {:10}  {:9}  {:40}  {:40}  {:2}
{:15}'.format(str(elem[0]),str(elem[1]),str(elem[2]),str(elem[4])+',
'+str(elem[3]),str(elem[5]),str(elem[6])))
```

```
                                    longitud = len(resultado)
                                    for elem in resultado:
                                          indice = resultado.index(elem)
                                          for ref in range(indice,longitud):
                                                if elem[0] in resultado[ref] and elem[1] in
resultado[ref] and indice != ref and elem[4] not in resultado[ref]:
                                                      print('Hay dos profesores diferentes que
aparecen en el aula seleccionada para el mismo intervalo de tiempo:\n')
                                                      print(' {:10}  {:9}  {:40}  {:40}  {:2}
{:15}'.format(str(elem[0]),str(elem[1]),str(elem[2]),str(elem[4])+',
'+str(elem[3]),str(elem[5]),str(elem[6])))
                                                      print(' {:10}  {:9}  {:40}  {:40}  {:2}
{:15}'.format(str(resultado[ref][0]),str(resultado[ref][1]),str(resultado[ref][2]),str(resulta
do[ref][4])+', '+str(resultado[ref][3]),str(resultado[ref][5]),str(resultado[ref][6])))
                                    input()


                              elif eleccion2 == '0':
                                    break
                        conexion.commit()
                        conexion.close()



      elif eleccion0 == '3':
            while True:
                  import sqlite3
                  conexion = sqlite3.connect("sabana.db")
                  cursor = conexion.cursor()
                  system('cls')
                  print('\n MODIFICAR BASE DE DATOS\n\n 1 - Asignaturas\n 2 - Profesores\n 3
- Grupos\n 4 - Aulas\n 5 - Sesiones\n 0 - Atrás')
                  eleccion3 = input('\n Elija una tabla: ')
                  if eleccion3 == '1':
                        tabla = 'asignaturas'
                  elif eleccion3 == '2':
                        tabla = 'profesores'
                  elif eleccion3 == '3':
                        tabla = 'grupos'
                  elif eleccion3 == '4':
                        tabla = 'aulas'
                  elif eleccion3 == '5':
                        tabla = 'sesiones'
                        system('cls')
                        fecha_mod = input('\n Introduzca el día en el cual quiere realizar
la modificación (AAAA-MM-DD): ')
                        if fecha_mod == '0':
                              continue
                  elif eleccion3 == '0':
                        break
                  else:
                        continue
                  system('cls')
                  print('\n TABLA', tabla.upper(), '\n\n 1 - Modificar entrada\n 2 - Añadir
entrada\n 3 - Eliminar entrada\n 0 - Atrás')
                  eleccion3x = input('\n Elija una opción: ')
                  system('cls')
                  print('')

                  if eleccion3x == '1':
                        if eleccion3 == '5':
                              cursor.execute("SELECT * FROM "+tabla+" WHERE fecha =
'"+fecha_mod+"'")
                        else:
```

```
                              cursor.execute("SELECT * FROM "+tabla)
                        titulos = []
                        for col in cursor.description:
                              titulos.append(col[0].upper())
                        resultado = cursor.fetchall()
                        columnas = len(resultado[0])
                        esqueleto = ' {:>13} '+' {:<20}'*(columnas-1)
                        print(esqueleto.format(*titulos),'\n')
                        for tupla in resultado:
                              print(esqueleto.format(*tupla))
                        eleccion3x_1 = input('\n Elija línea a modificar: ')
                        if eleccion3x_1 == '0':
                              continue
                        print('\n')
                        for titulo in titulos:
                              indice = titulos.index(titulo)
                              columna = titulo.upper()
                              print(' '+str(indice+1),'->',columna)
                        eleccion3x_2 = int(input('\n Elija el parámetro a modificar: '))
                        if eleccion3x_2 == 0:
                              continue
                        parametro = titulos[eleccion3x_2-1]
                        print('\n MODIFICAR',parametro.upper())
                        eleccion3x_3 = input('\n Escriba el valor deseado: ')
                        cursor.execute("SELECT * FROM "+tabla+" WHERE id"+tabla+" =
"+eleccion3x_1)
                        tupla = cursor.fetchone()
                        mod = list(tupla)
                        mod[eleccion3x_2-1] = eleccion3x_3
                        tuplam = tuple(mod)
                        while True:
                              eleccion3x_2c = input('\n ¿Desea realizar esta
modificación?\n\n Antigua entrada:\n'+esqueleto.format(*tupla)+'\n\n Entrada
modificada:\n'+esqueleto.format(*tuplam)+'\n\n 1 -> Sí\n 0 -> No\n  ')
                              if eleccion3x_2c == '1':
                                    cursor.execute("UPDATE "+tabla+" SET "+parametro+" =
'"+eleccion3x_3+"' WHERE "+titulos[0]+" = "+eleccion3x_1)
                                    print('\n La entrada fue modificada.')
                                    input()
                                    break
                              elif eleccion3x_2c == '0':
                                    break

                  elif eleccion3x == '2':
                        print('\n AÑADIR A '+tabla.upper()+'\n')
                        cursor.execute("SELECT * FROM "+tabla+" ORDER BY id"+tabla+" DESC
LIMIT 1")
                        ultimo = cursor.fetchone()
                        columnas = len(ultimo)
                        esqueleto = ' {:>13} '+' {:<20}'*(columnas-1)
                        titulos = []
                        for col in cursor.description:
                              titulos.append(col[0].upper())
                        tupla = []
                        for titulo in titulos:
                              if titulos.index(titulo) == 0:
                                    tupla.append(str(ultimo[0]+1))
                                    continue
                              tupla.append(input(titulo.upper()+': '))
                        tupla = tuple(tupla)
                        while True:
                              eleccion3x_c = input('\n ¿Desea añadir esta
entrada?\n\n'+esqueleto.format(*tupla)+'\n\n 1 -> Sí\n 0 -> No\n  ')
```

```
                                    if eleccion3x_c == '1':
                                          cursor.execute("INSERT INTO "+tabla+" VALUES
"+str(tupla))
                                          print('\n La entrada fue añadida.')
                                          input()
                                          break
                                    elif eleccion3x_c == '0':
                                          break


                        elif eleccion3x == '3':
                              if eleccion3 == '5':
                                    cursor.execute("SELECT * FROM "+tabla+" WHERE fecha =
'"+fecha_mod+"'")
                              else:
                                    cursor.execute("SELECT * FROM "+tabla)
                              titulos = []
                              for col in cursor.description:
                                    titulos.append(col[0].upper())
                              resultado = cursor.fetchall()
                              columnas = len(resultado[0])
                              esqueleto = ' {:>13} '+' {:<20}'*(columnas-1)
                              print(esqueleto.format(*titulos),'\n')
                              for tupla in resultado:
                                    print(esqueleto.format(*tupla))
                              eleccion3x_4 = input('\n Elija línea a eliminar: ')
                              if eleccion3x_4 == '0':
                                    continue
                              cursor.execute("SELECT * FROM "+tabla+" WHERE id"+tabla+" =
"+eleccion3x_4)
                              tupla = cursor.fetchone()
                              while True:
                                    eleccion3x_4c = input('\n ¿Desea eliminar esta
entrada?\n\n'+esqueleto.format(*tupla)+'\n\n 1 -> Sí\n 0 -> No\n  ')
                                    if eleccion3x_4c == '1':
                                          cursor.execute("DELETE FROM "+tabla+" WHERE
id"+tabla+" = "+eleccion3x_4)
                                          print('\n La entrada fue eliminada.')
                                          input()
                                          break
                                    elif eleccion3x_4c == '0':
                                          break



                        elif eleccion3x == '0':
                              continue
                        conexion.commit()
                        conexion.close()



            elif eleccion0 == '4':
                  system('cls')
                  import sqlite3
                  conexion = sqlite3.connect("sabana.db")
                  cursor = conexion.cursor()
                  while True:
                        orden = input('(0 -> Volver) SQL: ')
                        if orden == '0':
                              break
                        try:
                              cursor.execute(orden)
```

```
                except:
                        print(' Comando no válido.')
                        continue
                if 'SELECT' in orden or 'select' in orden:
                        titulos = []
                        for col in cursor.description:
                                titulos.append(col[0].upper())
                        resultado = cursor.fetchall()
                        if resultado == []:
                                print(' Ninguna coincidencia.')
                                continue
                        columnas = len(resultado[0])
                        esqueleto = ' {:<19}'*columnas
                        print(esqueleto.format(*titulos),'\n')
                        for tupla in resultado:
                                print(esqueleto.format(*tupla))
        conexion.commit()
        conexion.close()


elif eleccion0 == '0':
        break
system('cls')
```