UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DISEÑO DE TERMINAL PARA CONEXIÓN DESDE EL LABORATORIO DE INGENIERÍA ELÉCTRICA A UPVNET BASADO EN RASPEBRRY PI

AUTOR: KADI HANIFI LEBTAHI, NADIM

TUTOR: ROLDÁN PORTA, CARLOS

COTUTOR: ROLDÁN BLAY, CARLOS

Curso Académico: 2015-16

RESUMEN

A continuación se presentan una serie de documentos realizados para el diseño de un terminal para conectarse a UPVNET desde el laboratorio de Ingeniería Eléctrica, principalmente para realizar los test de la asignatura, basado en Raspberry Pi. En la memoria descriptiva se explica detalladamente el hardware y el software utilizado, así como el lenguaje de programación Python, base del proyecto desarrollado. Una vez hecho esto se procederá a explicar el código y se evaluarán otras alternativas que ofrece la Raspberry Pi en el laboratorio, antes de finalizar la memoria con una conclusión a modo de balance del trabajo realizado. Aparte de la memoria, los otros documentos son una valoración económica, un plano y los anexos a la memoria.

Palabras clave: Raspberry Pi, Python, terminal, UPVNET, Ingeniería Eléctrica, laboratorio.

ÍNDICE GENERAL DEL TRABAJO

MEMORIA

Nadim Kadi Hanifi Lebtahi

ÍNDICE

1. Introducción

1.1. Objetivos

El objetivo general de este Trabajo de Fin de Grado es diseñar un terminal interactivo que permita acceder a las aplicaciones de UPVNET desde el microordenador conocido como Raspberry Pi. Se debe diseñar tanto el apartado gráfico del terminal, como la funcionalidad. Para ello, se empleará el lenguaje de programación Python. Más específicamente, el objetivo principal es que la aplicación desarrollada permita acceder a los tests de evaluación de Poliformat, que se suelen realizar en el laboratorio de Ingeniería Eléctrica después de una práctica.

Además, se estudiarán otras posibilidades ofrecidas por la Raspberry Pi y se plantearán aplicaciones que podrían implementarse en un futuro para el laboratorio de Ingeniería Eléctrica.

Para el desarrollo de este TFG, se emplean conocimientos adquiridos a lo largo de la carrera. Por ejemplo, se revisan conceptos de computadores adquiridos en la asignatura de Tecnología Informática Industrial, así como conceptos de programación estudiados en esa misma asignatura o Informática de primero. Asimismo, para la elaboración del presupuesto se aplican procedimientos vistos en la asignatura de Proyectos, y para los planos se han empleado conocimientos de Ingeniería Gráfica.

Se ha incidido también en el aprendizaje a desarrollar de forma autónoma. Se ha debido investigar la instalación, configuración, programación y funcionamiento de la Raspberry Pi, así como se han estudiado las posibilidades que esta ofrece, intentando enfocarlo lo máximo posible a lo que requiere el proyecto. También se ha debido aprender el lenguaje de programación Python, y las cosas que se pueden desarrollar con él, ya que en el poco temario programación que se estudia en la carrera no está incluido este lenguaje.

Todo esto nos deja un TFG muy interesante, que mezcla conceptos ya conocidos, con otras áreas de estudio menos habituales en un grado como el Grado en Ingeniería en Tecnologías Industriales.

1.2. Motivación

La necesidad de este proyecto surge del Departamento de Ingeniería Eléctrica. En las asignaturas de este Departamento - como Teoría de Circuitos, Máquinas Eléctricas o Tecnología Eléctrica – es habitual realizar un test, extraído de Poliformat, al acabar una práctica, para evaluar los conocimientos adquiridos a lo largo de la misma. Actualmente, estos test se realizan bien en un PC, bien en una Tablet. Por ello, se ha decidido evaluar la posibilidad de emplear una Raspberry Pi, en lugar de los aparatos mencionados anteriormente. Esto podría suponer una reducción del espacio necesario y, sobre todo, un considerable ahorro económico, debido a que este microordenador tiene un coste muy reducido, y sólo sería necesario añadirle algún hardware de coste razonable.

A parte de esta necesidad particular, también se ha considerado interesante estudiar a qué otras funciones de UPVNET de las que sería interesante disponer en el laboratorio de prácticas es posible acceder desde el terminal, además de otros usos que se le puede dar a la placa e integrarla en el laboratorio. Con este proyecto, se contribuye a abrir un poco más el camino a un posible uso habitual de las Raspberry Pi en el futuro en la universidad, para resolver las necesidades que puedan surgir.

A título personal, mi principal motivación a la hora de elegir este TFG ha sido el interés en profundizar en el aprendizaje de la programación, una disciplina que se estudia muy poco en nuestra carrera. También estaba interesado en conocer más a fondo las posibilidades que ofrece la Raspberry Pi, un microordenador del que ya me había informado anteriormente y que ofrece diversas funcionalidades. Por último, mencionar que el hecho de que gran parte del aprendizaje sea autónomo y que requiera una gran cantidad de investigación por mi propia cuenta propia me ha parecido un reto muy interesante.

1.3. Estructura del TFG

El proyecto se divide en cuatro grandes partes: la memoria, los planos, el presupuesto y los anexos.

La memoria comienza con la presente introducción, donde se informa de los objetivos, la motivación y la estructura del Trabajo.

A continuación, se pasa a hablar de la tecnología utilizada para el desarrollo del proyecto. Este apartado está dividido en dos: Hardware y Software. En el primero, se habla de la placa Raspberry Pi: de qué componentes está formada, qué entradas y salidas tiene, precio, distintos modelos, aplicaciones realizadas con ella, etc. También se mencionan los dispositivos de entrada/salida utilizados. En el segundo sub-apartado, se habla del aspecto intangible necesario para el funcionamiento de la Raspberry y nuestra aplicación. Se empieza hablando de la historia del sistema operativo empleado y de sus principales características, pasando luego a explicar la configuración de la Raspberry Pi y a comentar alguna particularidad interesante. También se menciona algún software interesante, como los programas utilizados para el acceso remoto. Por último, se trata el lenguaje de programación conocido como Python, que representa la base de nuestro trabajo.

A continuación, se pasa a hablar de la programación, las líneas de programa escritas en lenguaje Python, para el desarrollo del terminal. Se ha dividido en dos partes bien diferenciadas, con la intención de que quede lo más claro posible: la primera corresponde a la parte gráfica de la aplicación, realizada con la librería gráfica Tkinter, mientras que la segunda abarca la funcionalidad de la aplicación. Ambas partes tienen una breve explicación de las librerías utilizadas y una explicación concienzuda del código, acompañada de capturas de pantalla.

Después de esto, hemos incluido un breve manual de usuario que pretende informar de cómo se debe usar la aplicación, aunque ésta se ha hecho de forma que sea lo más sencilla y explicativa posible, por lo que no debería haber problemas en este aspecto.

Una vez hemos terminado de hablar del terminal diseñado, pasamos a estudiar los pins de entradas y salidas digitales que presenta la Raspberry Pi y las posibilidades que estos ofrecen. También propondremos unas cuantas aplicaciones realizables con la placa y de las que podría ser interesante disponer en el laboratorio de Ingeniería Eléctrica.

Para ir finalizando con la memoria, se ha dejado un apartado de conclusiones, donde se ha incluido, por una parte, tanto la evaluación del trabajo realizado como la evaluación de los resultados y, por otra parte, la valoración personal que el alumno hace de este Trabajo.

Terminando con la memoria, tenemos la bibliografía, como debe poseer cualquier trabajo que se precie.

Después de la memoria, nos encontramos con el presupuesto. En él se ha incluido el precio de los componentes tecnológicos empleados y el sueldo que correspondería al desarrollador del proyecto, para después comparar cualitativamente con los métodos usados actualmente y dilucidar si el uso de estos terminales resultaría rentable.

Una vez visto el presupuesto, pasamos a los planos. Este no es un TFG que requiera una gran cantidad de ellos, pero se ha juzgado interesante incluir un dibujo de la placa y sus conexiones, que permitirá ver más claramente su funcionamiento.

Por último, hemos incluido un apartado de anexos, en el que va recogido todo el código en lenguaje Python que ha sido necesario para realizar el proyecto. También hemos incluido enlaces a todo el Hardware utilizado, o que hemos decidido que podría utilizarse en caso de implementación.

2. Tecnología empleada en el proyecto

2.1. Hardware

2.1.1. Raspberry Pi

2.1.1.1. ¿Qué es la Raspberry Pi?

El hardware principal con el que se ha trabajado para el desarrollo del proyecto es la Raspberry Pi. Una Raspberry Pi es una microcomputadora, o una SBC – *Single Board Computer*, lo que en castellano se traduciría como Ordenador de Placa Reducida. La característica principal de estas SBC es que todos los elementos necesarios para el funcionamiento del computador, que serían el procesador, la memoria RAM y los controladores de Entrada y Salida con los respectivos puertos USB, se encuentran impresos en una única placa base.

Los primeros microcomputadores surgieron en la década de los 70, con sistemas como el Datapoint 2200 o el Kenbak-1. Desde los años 1980, el término microcomputador ha caído en desuso. En aquella época, una de las características de lo que se conocía como "microcomputador" era el uso de microprocesadores, algo que actualmente se puede encontrar en gran cantidad de dispositivos. Por aquel entonces, ese término servía para diferenciar los ordenadores desarrollados para un uso personal de los grandes computadores, lo que ahora se puede sustituir por el término PC, Personal Computer, que incluiría por ejemplo los ordenadores de sobremesa. Por este motivo, de ahora en adelante, trataremos a la Raspberry Pi como "SBC".

Los ordenadores de placa reducida son posibles gracias a una elevada densidad de circuitos en la placa. Esto permite tener, en un especio muy reducido, prestaciones no tan alejadas a las de otros computadores no-SBC, puesto que el hecho de tener todos sus componentes integrados en la placa base no le impide poseer un procesador potente o una memoria RAM elevada. Además, este tipo de ordenadores pueden suponer un ahorro económico considerable. Son, en general, más baratos que sus homólogos lo que, añadido al hecho de que las placas base se suelen fabricar en grandes cantidades, permite ahorrar a gran escala. Gracias a esto, se puede hacer prácticamente lo mismo que en un PC, a un menor coste. También permite el acceso a la informática en zonas más empobrecidas del planeta. Las SBC tienen aplicaciones interesantes y conviene tenerlas en cuenta.

Sin embargo, también se deben mencionar los inconvenientes de las SBC. El principal problema que se deriva del uso de estos ordenadores es que, en general, no se pueden reparar ni actualizar, exceptuando casos especiales (como por ejemplo si el problema se pudiera solucionar soldando un componente a la placa base). La solución más habitual es reemplazar la SBC. Debemos tener en cuenta que el software avanza muy rápidamente, y para poder aprovechar este desarrollo necesitamos un hardware que le acompañe. Por eso, sería necesario actualizar las placas y presentar nuevas versiones regularmente, para evitar que queden obsoletas con el tiempo. También cabe aludir a los periféricos necesarios para el aprovechamiento total de estos ordenadores. El hecho de necesitar usar un monitor, teclado, ratón… puede acabar encareciendo el producto.

Aparte de la Raspberry Pi, uno de los ordenadores de placa reducida más conocidos es la Placa Arduino (Fig. 1), que está principalmente enfocada al desarrollo de proyectos relacionados con la electrónica, mediante el uso de actuadores y sensores. Asimismo podemos mencionar otros SBC como el MK802 (Fig.2) – el modelo de Android, con un diseño elegante y muy buenas prestaciones – o el BeagleBoard.

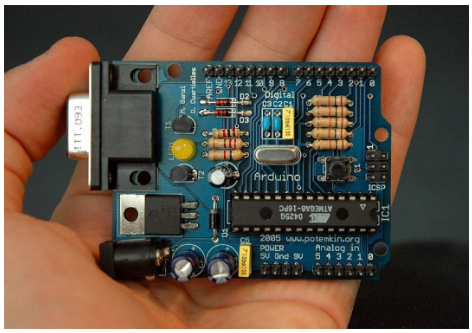


Fig. 1: Placa Arduino RS232

Fig. 2: SBC Android MK802

Volviendo a lo que nos concierne, hablemos del ordenador de placa reducida que nos interesa: la Raspberry Pi. Este ordenador fue desarrollado en el Reino Unido por la Fundación Raspberry Pi. El objetivo de dicha Fundación, creada en 2009, es el de fomentar el estudio de las ciencias de la computación en las escuelas, de ahí su necesidad de crear la placa Raspberry Pi. Su intención es que, desde muy jóvenes, los niños puedan adentrarse en el mundo de la programación, con un ordenador simple, accesible y que se puede obtener a un precio asequible. Los primeros modelos Alpha y Beta aparecieron en 2011, mientras que el primer modelo que se comercializó a larga escala, la Raspberry Pi A (Fig. 3), surgió en 2012. En 2016, la Fundación Raspberry Pi anunció que se habían vendido 8 millones de ejemplares, convirtiéndose en el ordenador para uso personal más vendido de la historia en Reino Unido.

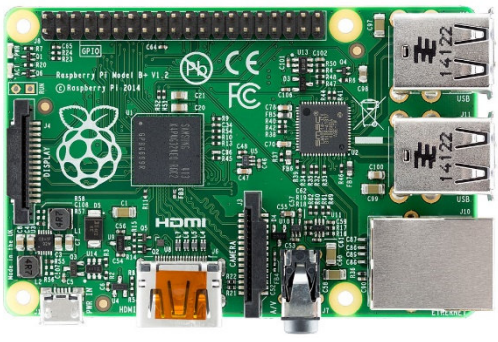


Fig. 3: Raspberry Pi Modelo A

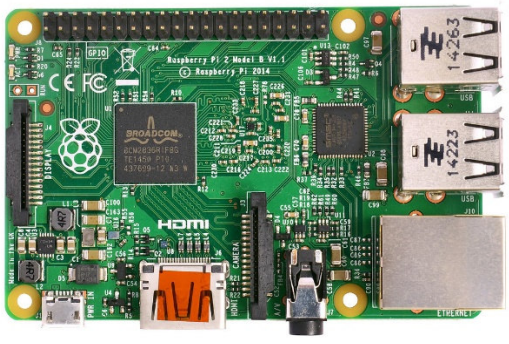


Fig. 4: Raspberry Pi 2 Modelo B

Desde el desarrollo de la primera placa en 2012, la Raspberry Pi se ha visto forzada a ir actualizándose, siguiendo la línea de lo que se ha comentado más arriba, donde se establecía que para poder seguir el desarrollo constante del software el hardware debía adaptarse. Por ejemplo, el primer modelo tenía 256 MB de Memoria RAM y solo un puerto USB, mientras que los modelos más actuales, como por ejemplo la Raspberry Pi 3, la más reciente a día de hoy, tienen hasta 4 veces más.

En este proyecto se ha utilizado la Raspberry Pi 2 Modelo B (Fig. 4). Esta versión fue lanzada en febrero de 2015 a un precio de salida de 35$ (unos 31€), siendo la sucesora de la Raspberry Pi 1 Modelo B+. Tiene mejoras significativas con respecto a esta, como podemos ver en su tabla de características, extraídas de la página oficial de Raspberry Pi (www.raspberrypi.org/products/raspberry-pi-2-model-b/) y de la página correspondiente de Wikipedia (https://es.wikipedia.org/wiki/Raspberry_Pi):

Modelo	Raspberry Pi 1 Modelo A	Raspberry Pi 1 Modelo B+	Raspberry Pi 2 Modelo B
SoC	Broadcom 2835		Broadcom 2836
CPU	700 MHz single-core ARM1176JZF-S		900 MHz quad-core ARM Cortex A7
GPU	Broadcom VideoCore IV,60 OpenGL ES 2.0, MPEG-2 y VC-1, 1080p30		
Puertos USB	1	4	4
Memoria RAM	256 MB	512 MB	1 GB
Entradas de vídeo	Conector MIPI CSI que permite instalar un módulo de cámara desarrollado por la Raspberry Pi Foundation		
Salidas de vídeo	Conector RCA (PAL y NTSC), HDMI, Interfaz DSI para panel LCD		
Salidas de audio	Conector de 3.5 mm, HDMI		
Almacenamiento	SD	Micro SD	
Conectividad de RED	-	Ethernet	
Dimensiones / Peso	85.60mm × 53.98mm / 45 g		
Consumo Energético	500mA (2.5 W)	600mA (3 W)	800mA (4 W)
Fuente de alimentación	5 V vía Micro USB o GPIO header		

Tabla 1: Caracteríticas y especificaciones técnicas de la Raspberry Pi

2.1.1.2. CPU: Arquitectura y características

El SoC se refiere a *System-on-a-chip*, la placa donde están integrados la mayor parte de los componentes del sistema, incluyendo la CPU y la GPU. En cuanto al procesador, podemos ver que se ha producido una mejora considerable, pasando de un simple-core de 700 MHz, a un quad-core de 900 MHz ARM. La memoria RAM ha aumentado hasta 1 GB. Estas características son más que decentes para un ordenador de este tamaño y con este precio tan reducido, acercándose a las de algunos *smartphones* recientes.

La CPU de la Raspberry Pi tiene una arquitectura ARM. La arquitectura de un procesador se refiere al conjunto de instrucciones entendibles y ejecutables por la CPU. También se le puede llamar juego de instrucciones o ISA (del inglés *instruction set architecture*). Además del repertorio de instrucciones, el concepto de "arquitectura" también incluye los registros, modelo de excepciones, manejo de la memoria virtual, mapa de direcciones físicas y otras características. Resumiendo, la arquitectura incluye todo aquello que el programador debe saber acerca de la máquina.

Los dos tipos de arquitectura de microprocesadores principales son la arquitectura CISC (*Complex Instruction Set Computer*) y la arquitectura RISC (*Reduced Instruction Set Computer*).

El primer tipo se caracteriza por un conjunto de instrucciones muy extenso, que permite realizar operaciones complejas en la memoria o en los registros internos. El inconveniente de este tipo de arquitectura es que el procesamiento de tareas en paralelo – es decir, la ejecución de instrucciones simultáneamente – se vuelve complicado. Este tipo de arquitectura es la "clásica", que parte de la creencia de que instrucciones más completas darán lugar a un procesador más potente, un acercamiento que se está empezando a desestimar hoy en día. Aun así, sigue siendo el ISA más usado actualmente en los ordenadores portátiles, siendo la base de los procesadores Intel x86 y AMD x86-84.

En oposición a la arquitectura CISC, encontramos la RISC, usada en microprocesadores que poseen un conjunto de instrucciones presentadas en un número de formatos reducido y que tienen un acceso más restringido a la memoria. Gracias a la arquitectura RISC, se pueden ejecutar instrucciones paralelas y segmentadas, lo que permite que se ejecuten instrucciones simultáneas y que una nueva etapa pueda iniciarse sin que hayan acabado las acciones de la etapa anterior, lo que aumenta considerablemente la velocidad. La filosofía de desarrollo de procesadores RISC parte de una nueva visión muy distinta a la que se tenía con la arquitectura CISC, que establece que un procesador más potente debería permitir emplear instrucciones más sencillas, lo que a su vez dará lugar a una mayor velocidad.

Los procesadores RISC todavía no han conseguido desbancar a los CISC, que como hemos mencionado se siguen utilizando en los ordenadores portátiles, aunque se puede ver claramente como su popularidad va en aumento. Sin ir más lejos, las versiones más recientes de procesadores x86 traducen las instrucciones basadas en CISC a instrucciones basadas en RISC, de tratamiento más sencillo, para que se puedan usar internamente, antes de ejecutarlas. Además, se usan procesadores de tipo RISC para muchas otras aplicaciones. Por ejemplo, una familia de microprocesadores que emplean la arquitectura RISC, llamada MIPS, se utiliza para usos variados tales como sistemas embebidos, routers Cisco, videoconsolas como la Nintendo 64 o la PlayStation e incluso en la sonda New Horizons de la NASA. También cabe mencionar los procesadores ARM, utilizados en videoconsolas como la Nintendo DS, productos como el iPad y, por supuesto, en la Raspberry Pi.

ARM es una familia de procesadores con arquitectura RISC. Con estos procesadores se pueden aprovechar las ventajas que tienen los RISC con respecto a los CISC, a saber, el uso de instrucciones más sencillas, para que se puedan ejecutar simultáneamente, además de emplear una cantidad mucho menor de transistores, lo que reduce tanto los costes como el calentamiento del microprocesador. En la actualidad, el diseño de ARM es uno de los más utilizados en el mundo. Se estima que 3 de cada 4 procesadores de 32 bits poseen un chip ARM. Su uso se centra principalmente en dispositivos móviles, como smartphones o tablets, más que en ordenadores de escritorio. El uso de esta arquitectura ARM explica por qué la Raspberry Pi es capaz de funcionar solo con una fuente de alimentación de 5V, y además no requiere un disipador térmico. Como inconveniente al uso de la arquitectura ARM, podríamos mencionar que la Raspberry Pi en general no será compatible con el software usado por los PC tradicionales.

Más concretamente, la Raspberry Pi usa un procesador de la familia ARM-v7. Esta familia presenta una importante mejora con respecto a sus predecesores en lo que respecta a los buses

de procesamiento de instrucciones. También puede soportar mayores velocidades de reloj –puede aguantar un overclock a 1GHz –, aumentando la rapidez del procesamiento y la simultaneidad. El núcleo usado por la Raspberry es el ARM Cortex-A7. Es similar al usado por algunos modelos de iPhone y iPod Touch de hace unos años. El chip empleado, como se puede ver en la Tabla 1, es un Broadcom 2836. Ha necesitado ser actualizado desde el Broadcom 2835 para admitir el Cortex-A7, ya que el anterior sólo podía emplear hasta un ARM11, lo que limitaba sus capacidades.

2.1.1.3. Otros elementos de la Raspberry Pi. Distribución

En la Fig. 5 podemos ver la distribución de los distintos elementos de la placa:

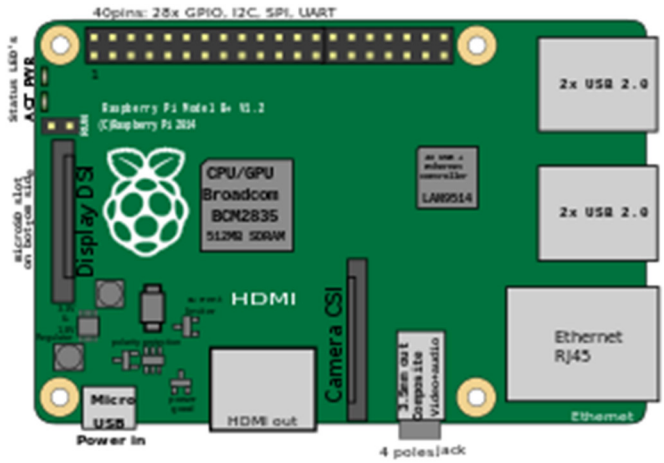


Fig. 5: Raspberry Pi 2 Modelo B. Elementos de la placa

En el centro encontramos el chip Broadcom, donde se encuentran la CPU y la GPU, justo en el lado opuesto de la memoria RAM, situada en la cara inferior. Del procesador ya hemos hablado bastante. En cuanto a la GPU, mencionar que permite reproducir Blu-Ray, dispone un núcleo 3D y es capaz de decodificar hasta 1080p30.

La Raspberry Pi no tiene disco duro ni disco de estado sólido. El almacenamiento permanente se debe hacer mediante una tarjeta micro SD, introducida en la parte izquierda de la imagen, dónde se deberá instalar el Sistema Operativo. En la parte derecha podemos encontrar 4 puertos USB, que nos deberían permitir conectar un número suficiente de periféricos. También podemos encontrar un puerto Ethernet, que nos permitirá conectar la placa directamente al router, para tener conexión a Internet.

Conviene señalar que la última versión, la Raspberry Pi 3, además del puerto Ethernet, está equipada con 2.4 GHz WiFi 802.11n (600 Mbit/s) y Bluetooth 4.1 (24 Mbit/s), algo que no está incluido en la placa usada en este TFG. En nuestro caso, también podríamos acceder a internet conectando la placa directamente al ordenador portátil, o accediendo a la conexión Wi-Fi mediante un adaptador. Sin embargo, para el desarrollo del terminal se ha optado por la conexión directa al router por cable Ethernet.

Siguiendo con el repaso de los elementos de la placa, tenemos en la parte inferior el conector de alimentación, que requiere una fuente de alimentación de 5 voltios. También vemos el

conector HDMI, que nos permitirá conectar la Raspberry Pi fácilmente a un monitor o a una televisión, y el jack de audio, que funciona como salida de audio. Entre el Jack y el HDMI está el conector al que se puede incorporar la Pi NoIR Camera, cámara específica de la Raspberry Pi.

Aparte de todo esto, hay entradas y salidas digitales en los pins GPIO situados en la parte superior de la imagen. Hablaremos un poco más de ellos más adelante, pero interesa saber que permiten aumentar la funcionalidad de la placa o emplearla para otras aplicaciones, principalmente enfocadas a la electrónica.

A la izquierda de la placa se encuentra el display DSI, que permite conectar una pantalla LCD. También cabe mencionar los leds de estado, en la esquina superior izquierda, que indican si la Raspberry Pi está conectada a la fuente de alimentación y si está en funcionamiento.

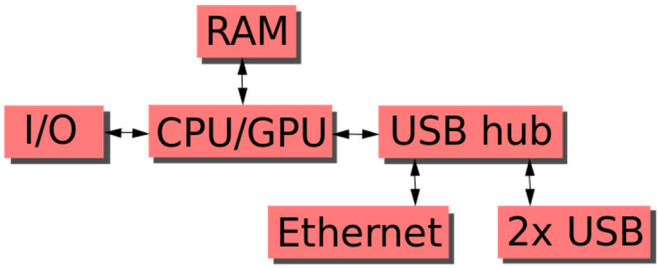


Fig. 6: Diagrama de bloques del modelo B

En el diagrama de bloques de la Fig. 6, podemos apreciar de forma esquemática las conexiones entre los distintos elementos de la Raspberry Pi modelo B, para cualquier versión (la única diferencia es el número de USB). Como elemento central tenemos el SoC, que incluye la CPU y la GPU. A él están conectados la memoria RAM, las entradas y salidas – digitales, de audio, de vídeo… - y los puertos USB – Ethernet y otros USB donde se pueden conectar los periféricos.

2.1.1.4. Aplicaciones prácticas de la Raspberry Pi

En este apartado presentaremos una serie de aplicaciones realizadas con la Raspberry Pi, para mostrar la versatilidad que ofrece. Estos son sólo unos ejemplos de lo que se puede hacer con la Raspberry Pi, entre otras muchas aplicaciones que van desde crear un reproductor de música a aplicaciones domóticas para abrir la puerta del garaje o encender las luces de casa.

1) Usar la Raspberry Pi como ordenador: la primera aplicación posible de la Raspberry Pi es la que puede parecer más evidente, usarla como un ordenador. Con una conexión a internet, un monitor, un teclado y un ratón, la Raspberry Pi 2 ofrece las suficientes prestaciones para un uso del día a día.

2) Usar la Raspberry Pi como emisora FM (Fig. 7): usando un cable como antena y programando un script para ejecutar la reproducción de audio se puede hacer que una radio reciba la señal.

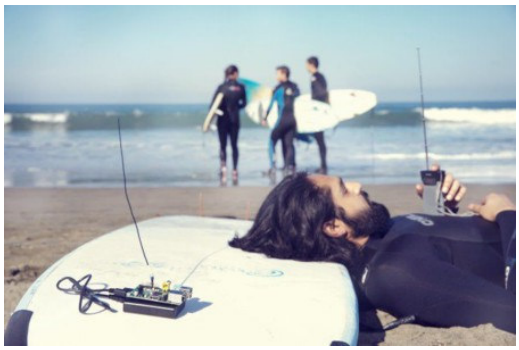


Fig. 7: Uso de la Raspberry Pi como emisora FM

3) Controlar la alimentación de tu mascota mediante la Raspberry Pi (Fig. 8): Un desarrollador llamado David Bryan ha creado un sistema para alimentar a sus gatos cuando se ausenta unos días.

Fig. 8: Dispensador automático de comida para gatos.

4) Usar la Raspberry Pi como videoconsola (Fig. 9): con una serie de botones y una pantalla se podría usar la Raspberry Pi como si de una consola se tratase.

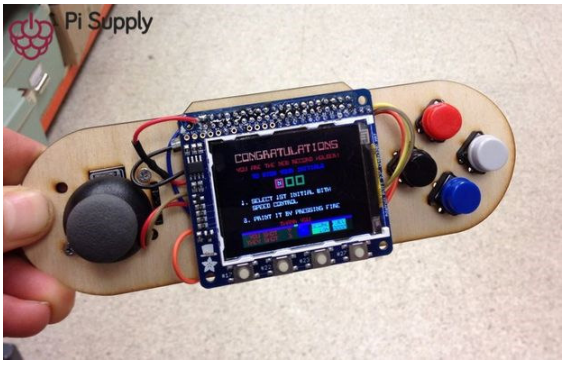


Fig. 9: Consola hecha con Raspberry Pi

Las funciones que puede tener la Raspberry Pi son muy variadas. No hay más límite que la imaginación.

2.1.2. Periféricos. Entradas y salidas

Además de la Raspberry Pi, nuestro computador, necesitaremos un hardware adicional.

En primer lugar, se necesita una tarjeta Micro SD, donde se instalará el Sistema Operativo, y que además actuará como memoria, ya que hemos mencionado que la Raspberry Pi no tiene disco duro ni disco de estado sólido. La tarjeta utilizada es una SanDisk SDSQUNC-016G-GZFMA microSDHC Clase 10 de 16 GB (Fig. 10). Esta capacidad es más que suficiente para la correcta instalación del sistema operativo y el funcionamiento de la Raspberry Pi. Más adelante se explicará cómo se puede "ampliar" esta capacidad para evitar que el espacio ocupado por el sistema sea limitante.

La alimentación de la Raspberry Pi, de 5 voltios, ya viene incluida en la caja de la Raspberry Pi. Consiste en un cargador de la marca Stontronics LTD (Fig. 11), similar a un cargador de teléfono móvil, con sus respectivos adaptadores para distintos tipos de enchufe.

Se ha usado un teclado Logitech K120 y un ratón Logitech Mouse M150 (Fig. 12), ambos conectados a dos de los puertos USB de la placa. Ambos estaban disponibles en casa. También se tiene que conectar un monitor mediante HDMI. Para el desarrollo del proyecto se ha usado un televisor de marca Saivod modelo LCD 632CI-E (Fig. 13) disponible en casa. Cualquier pantalla o monitor que tenga conexión HDMI vale para conectar a la Raspberry Pi, así como se le podría conectar una pantalla táctil, lo que nos evitaría emplear un teclado y un ratón.

Fig. 10: Tarjeta Micro SD

Fig. 11: Cargador y adaptadores

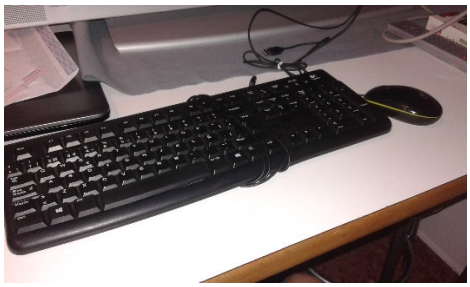


Fig. 12: Ratón y teclado

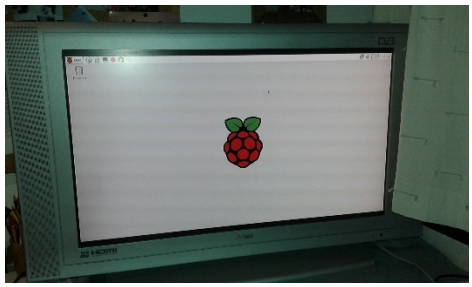


Fig. 13: Televisor conectado

Fig. 14: Raspberry Pi con todos los periféricos conectados

Más adelante, se verá una alternativa que requerirá simplemente que la Raspberry Pi esté conectada a la red eléctrica y a la red Ethernet, evitando el uso de los periféricos: el acceso por conexión remota desde un ordenador portátil.

_____

En el anexo aparecen links de enlace a todo el Hardware utilizado en el desarrollo del proyecto (aunque la información del televisor es algo escueta, ya que se trata de un modelo antiguo que ya no se comercializa), así como otras opciones más económicas que hemos valorado para su implementación a mayor escala.

2.2. Software

En este apartado, mencionaremos todo el software empleado en la Raspberry Pi, empezando por el sistema operativo, sus características y su instalación, siguiendo con la configuración de la placa y terminando con la explicación sobre el lenguaje de programación Python.

2.2.1. Sistema Operativo: Raspbian

El sistema operativo que emplea la Raspberry Pi es Raspbian. Este es una distribución del Sistema Operativo libre GNU/Lunix basado en Debian y desarrollado exclusivamente para la Raspberry Pi. Se procederá a explicar brevemente todos estos términos, que pueden no resultar muy comunes.

2.2.1.1 De Unix a Debian GNU/Linux

Unix (registrado como UNIX ®) es un sistema operativo desarrollado en 1969 por la corporación AT&T, entre cuyos desarrolladores destacan Ken Thompson y Dennis Ritchie. Unix es un sistema operativo portable – puede emplearse desde un USB, SD… sin necesidad de instalarlo –, multitarea – varias aplicaciones en ejecución simultáneamente – y multiusuario. Unix se escribió en el lenguaje de programación C, lo que supuso una revolución con respecto a los otros sistemas operativos en el mercado. Es un sistema interactivo y bastante flexible y elegante, puesto que las líneas de comando son relativamente breves y coherentes. sn potente, versátil y de fácil integración, y presenta gran facilidad para el direccionamiento de Entradas/Salidas.

Unix permite emplear una importante variedad de *shells.* Un *shell* es un intérprete de comandos, que conforma la parte esencial de la interfaz de usuario de los sistemas Unix o basados en Unix.

Básicamente, el *shell* es el intermediario entre el usuario y el sistema operativo. Debe leer los comandos, interpretarlos, ejecutar el comando y finalmente devolver el resultado. Los *shells* más importantesson sh (*Bourne shell),* bash *(Bourne again shell)* o csh *(C shell).* Cuando se necesita combinar comandos, se puede escribir en un fichero para ejecutarlo a continuación, lo que se conoce como *script.*

Ciertos sistemas operativos implementados a partir de UNIX han sido capaces de consolidarse en el mercado. Este es el caso de Solaris, desarrollado por Sun Microsystems, AIX, desarrollado por IBM, HP-UX, sistema operativo desarrollado por la famosa compañía Hewlett Packard, y, por supuesto, el entorno operativo OS X (antes conocido como Mac OS X), utilizado por los famosos ordenadores Macintosh de la compañía Apple Inc., y que desde su versión 10.5 cuenta con la certificación UNIX.

Sin embargo, existen sistemas operativos que no están certificados por UNIX ® y que sin embargo pueden incluir software libre y de código abierto, e incluso código fuente inspirado en Unix. Estos se suelen llamar sistemas operativos *Unix-like*, que en español podría traducirse por "como Unix" o "de tipo Unix", aunque a veces se abrevia como *nix o UN*x, para evitar problemas con la marca registrada. Uno de estos sistemas operativos Unix-like es GNU.

GNU (Fig. 15) es un sistema operativo desarrollado desde 1983 por Richard Stallman. Su nombre es un acrónimo recursivo, que significa *GNU's Not Unix* (GNU no es Unix, en español), lo que pretende señalar la diferencia que comentábamos arriba. El diseño de GNU está basado en Unix y es compatible que él, pero sin embargo no emplea código de Unix y es Software Libre. De hecho, el movimiento del software libre se inició con la creación de este sistema. A la hora de hablar de software libre debemos tener en cuenta la diferencia entre "libre" y "gratuito", que se puede confundir, principalmente en inglés con el término *free*; más que en el precio, debemos pensar en la libertad de expresión.

Citando al propio Richard Stallman (2004). Free software, free society: "un programa es software libre si los usuarios tienen las cuatro libertades esenciales:

- La libertad de ejecutar el programa como se desea, con cualquier propósito (libertad 0).
- La libertad de estudiar cómo funciona el programa, y cambiarlo para que haga lo que usted quiera (libertad 1). El acceso al código fuente es una condición necesaria para ello.
- La libertad de redistribuir copias para ayudar a su prójimo (libertad 2).
- La libertad de distribuir copias de sus versiones modificadas a terceros (libertad 3). Esto le permite ofrecer a toda la comunidad la oportunidad de beneficiarse de las modificaciones. El acceso al código fuente es una condición necesaria para ello."

Todas estas condiciones deben cumplirse para que los usuarios sean los que controlan el programa y no al revés, lo que sería injusto. Como vemos, la gratuidad o no del programa no resulta relevante a la hora de definir si un software es libre o no. La libertad de distribuir copias puede hacerse con o sin beneficio económico a cambio. Tampoco es relevante si el uso que se

le va a dar al programa es para el trabajo o como pasatiempo. En resumen, se podría decir que el software libre implica que nadie tiene "poder" sobre el programa. Así, el movimiento por el software libre promueve la solidaridad entre usuarios y desarrolladores.

GNU comparte las características fundamentales ya mencionadas de los sistemas Unix. La principal diferencia radica en el uso de software libre y de código fuente libre.

Fig. 15: Logo de GNU

En cuanto a Linux, es común confundirse y pensar que Linux se refiere exclusivamente al nombre del sistema operativo empleado en PCs o en la propia Raspberry Pi. Sin embargo, Linux no es únicamente un sistema operativo propiamente dicho, sino que se trata también de un núcleo o kernel, término usado habitualmente es informática. Un núcleo es un software esencial que forma parte del sistema operativo, pero no se trata del sistema operativo en sí. Las funciones principales de un núcleo son administrar la memoria para gestionar todos los programas en ejecución, acceder cómodamente a los periféricos y al Hardware y administrar el tiempo de procesador que requieren los programas en ejecución. También debe encargarse de gestionar las tareas y la comunicación entre los programas y el hardware.

Existen cuatro tipos de núcleos: núcleos monolíticos, micronúcleos, núcleos híbridos y exonúcleos (Fig. 16). Se hablará brevemente de los dos primeros, por ser los más importantes. Antes de proceder a esto, se debe tener claro el concepto "abstracción de hardware". La capa de abstracción de hardware (*Hardware abstracción layer, HAL,* en inglés) es un elemento del sistema que se encarga de la interacción entre el hardware y el software. Es una "capa" de software que interactuará con el hardware, haciendo que a su vez, cuando una aplicación requiera acceder a un hardware no lo hará directamente, sino que lo hará a través del HAL, haciéndolas independientes del hardware.

Los micronúcleos presentan abstracciones simples, para implementar servicios mínimos del sistema operativo como los espacios de direccionamiento, comunicación entre procesos y gestión de hilos. El resto de los servicios, como pueden ser las operaciones de Entrada/Salida o la gestión de la memoria, son ejecutados en modo de usuario, en vez de hacerlo a través el núcleo. Las principales ventajas de este tipo de núcleos son la mayor seguridad ante fallos, puesto que un fallo no comprometería todo el sistema, la reducción de la complejidad y una mayor facilidad a la hora de crear controladores de dispositivos. Sin embargo, la sincronización de los elementos del núcleo y el acceso a memoria se vuelven más complicados, además de que el rendimiento se ve comprometido. Un ejemplo de micronúcleo es Mach, el micronúcleo oficial de GNU, que se utiliza en Hurd.

En oposición, tenemos los núcleos monolíticos. Al contrario que en los micronúcleos, en los núcleos monolíticos todo el sistema operativo trabaja en el núcleo. Existe una única capa de abstracción de gran tamaño que implementa todos los servicios del sistema. La desventaja principal de este tipo de núcleo es que un único fallo en una rutina interna podría causar el fallo de todo el sistema. A cambio, presentan importantes ventajas, como por ejemplo un mayor rendimiento gracias a que sus sistemas están programados de forma no modular. Un ejemplo de núcleo monolítico es Linux, del que se hablará en los próximos párrafos.

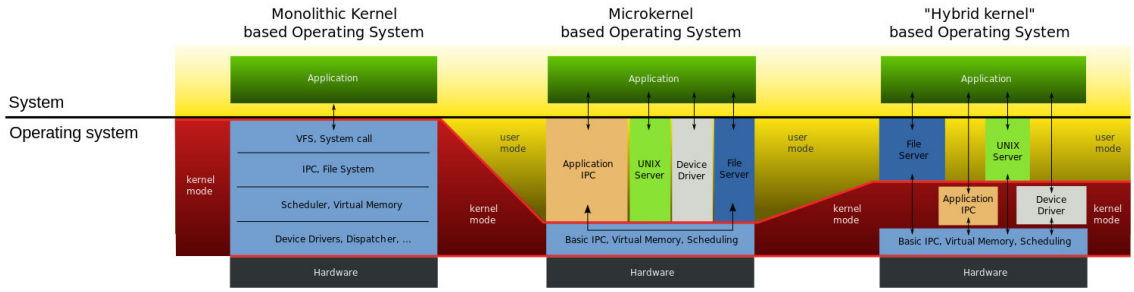


Fig. 16: Comparación entre los distintos tipos de núcleo

Así pues, Linux (Fig. 17) es un núcleo creado en 1991 por Linus Torvalds, de ahí su nombre. Linux se puede considerar a la vez tanto un sistema operativo como un núcleo. Esto es así porque en un sistema operativo monolítico, el núcleo y el SO se componen del mismo programa, al contrario de lo que sucede en los sistemas micronúcleo. Sin embargo, aun siendo un sistema con un núcleo monolítico, Linux tiene una peculiaridad: también es modular, lo que permite añadir o quitar componentes al núcleo incluso en pleno funcionamiento, previniendo que un error afecte a todo el sistema, solventando así uno de los inconvenientes de los núcleos monolíticos, pero sin perder gran parte del rendimiento que estos presentan. Aun así, en nuestro caso no nos interesa Linux como sistema operativo; nos interesa lo que llamamos distribución GNU/Linux, que emplea el kernel Linux junto con herramientas del sistema operativo GNU.

Fig. 17: Logo de Linux

Como ya hemos mencionado anteriormente, a menudo se llama a esta combinación como Linux, no diferenciando el núcleo del sistema operativo en el que se basa y las herramientas empleadas. Existe cierta controversia entre si la denominación correcta debería ser GNU/Linux o si simplemente bastaría con mencionar el nombre del núcleo. De hecho, en el desarrollo GNU se proyectaba también un núcleo, llamado Hurd, pero su desarrollo se paralizó al surgir Linux.

Los principales argumentos a favor de la denominación conjunta es evitar confusiones entre el núcleo y las herramientas, así como reconocer la labor de los desarrolladores de GNU al crear herramientas importantes para que el sistema operativo pueda ser compatible con Unix. Nosotros consideramos más justa esta denominación GNU/Linux, y es la que emplearemos.

Una de las principales características de GNU/Linux, que lo hace muy conocido y valorado por los consumidores, es la seguridad máxima. GNU/Linux es prácticamente inmune a los virus, debido a la "transparencia" del sistema, que dificulta mucho la creación de un programa que no sea visible. También es relevante la estabilidad del sistema, por lo que mencionábamos anteriormente del uso de un núcleo monolítico modular, evitando que un cuelgue o fallo pueda comprometer todo el sistema. Hay una gran variedad de distribuciones Linux, que se amoldan a todos los gustos, ya sea para usuarios habituales, seguridad, programadores, jugadores de videojuegos, etc. Además, la gran mayoría de los programas y aplicaciones son gratuitas (que no es lo mismo que libres, no lo olvidemos). Encontramos también una gran variedad de consolas virtuales. Por último, pero no menos importante, no olvidemos las propiedades de portabilidad, multitarea, multiusuario y multiplataforma que viene "arrastrando" desde Unix.

El sistema operativo Debian GNU/Linux (Fig. 18) surge como distribución de GNU/Linux por parte de una comunidad de desarrolladores y usuarios voluntarios, llamada Debian o Proyecto Debian, que colaboran a través de internet para desarrollar este software libre. Debian también desarrolla sistemas basados en GNU pero que emplean otros núcleos como Hurd, antes mencionado, o NetBSD. Proyecto Debian no vende su software, está creado libremente por usuarios y desarrolladores que lo ponen a disposición de cualquiera en la red, siguiendo las líneas del movimiento del software libre. Sin embargo, una empresa o particular puede comercializarlo, siempre y cuando respeten su licencia.

Además de tratarse de un software libre, Debian también presenta las características ya conocidas que hereda de GNU/Linux. Aunque se creó inicialmente para el procesador Intel 386, ahora está disponible para varias arquitecturas de computadoras: Pentium, AMD, Alpha, ARM – la correspondiente a la Raspberry Pi–, Motorola 68X y un largo etcétera. Hay una gran colección de software disponible: más de 43000 paquetes en la última versión. Lo acompaña una gran variedad de sistemas de gestión de paquetes, como dpkg, que permiten al usuario instalar, desinstalar y actualizar uno o varios paquetes. Se pueden instalar varios entornos gráficos diferentes, o incluso no instalar ninguno, ya que cuenta con un buen intérprete de órdenes que nos permitiría trabajar únicamente mediante comandos. La última versión disponible y estable es la 8.0, llamada Jessie, que fue lanzada en 2015.

Fig. 18: Logo de Debian

2.2.1.2 El sistema operativo Raspbian

Así, basado en Debian, surge la distribución Raspbian. Su nombre es una mezcla de Raspberry Pi + Debian (Fig. 19). Raspbian no está afiliado directamente con la Fundación Raspberry Pi, sino que fue creado por un grupo de desarrolladores que, según sus propias palabras, son "seguidores del hardware Raspberry Pi, los objetivos educacionales de la Fundación Raspberry Pi y, por supuesto, del proyecto Debian".

Fig. 19: Raspberry Pi + Debian = Raspbian

El sistema operativo es una adaptación no oficial de Debian a la CPU ARM empleada por la Raspberry Pi, ya que este sistema no tolera todas las arquitecturas de computadores. El soporte ha sido optimizado para hacer un uso importante cálculos en coma flotante, lo que permite aumentar el rendimiento.

Raspbian presenta un entorno gráfico llamado LXDE (*Lightweight X11 Desktop Environment*) (Fig. 20). Se trata de un entorno de código abierto diseñado para Unix y otras plataformas Unix-like como GNU/Linux, que empezó a desarrollarse en Taiwan en 2006. La idea era crear una interfaz ligera y barata, lo que permite ahorrar energía y aumentar la velocidad, proporcionado una rápida conexión con aplicaciones de red. Todo esto lo hace útil para procesadores de rendimiento relativamente bajo como el de la Raspberry Pi. El gestor de ventanas utilizado por LXDE, que controla ubicación y la apariencia de las ventanas, es Openbox.

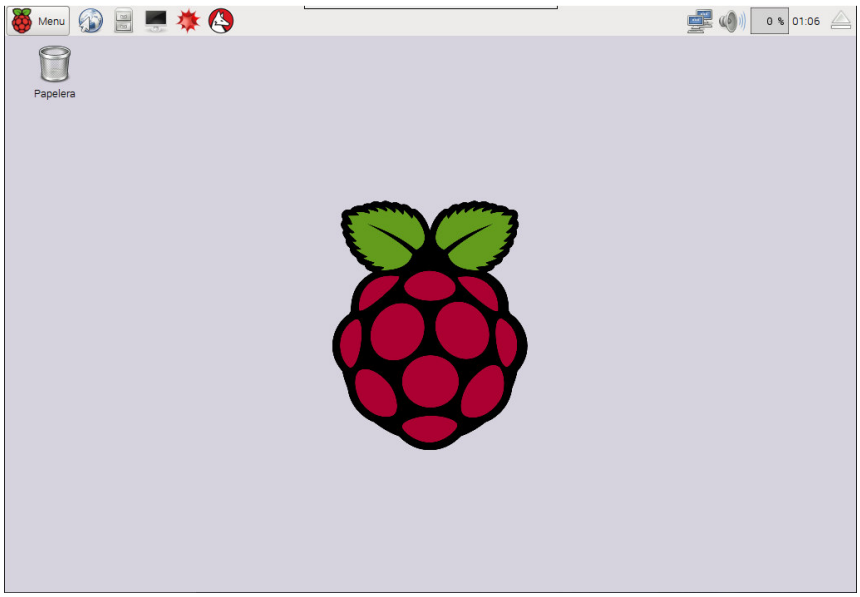


Fig. 20: Entorno gráfico de Raspbian

En la imagen anterior podemos ver el escritorio de raspbian. Los iconos en la esquina superior derecha nos muestran, respectivamente: la conexión a la red disponible actualmente, el

volumen, porcentaje de uso de la CPU, la hora y un botón que nos permitiría extraer de forma segura dispositivos conectados a la placa, como podría ser un USB.

En la esquina superior izquierda, justo a la derecha del menú, tenemos el icono del navegador web llamado Epiphany. A continuación vemos el explorador de archivos. Junto a él, tenemos lo que se llama LXTerminal, que se trata básicamente del *shell*, el intérprete de comandos de Raspbian. A la derecha, tenemos dos iconos: el primero nos llevará al programa Wolfram Mathematica – una versión piloto creada para la Raspberry Pi – y el segundo al intérprete de comandos de Wolfram.

Si desplegamos el Menú, podemos encontrar gran variedad de aplicaciones. Tenemos un primer apartado de programación, desde el que podemos acceder a varias aplicaciones para diversos lenguajes. Tenemos dos IDE (*Integrated Development Environment,* Entorno de Desarrollo Integrado) de Java: BlueJ – creado para el aprendizaje de Programación Orientada a Objetos – y Greenfoot – útil para desarrollar pequeños juegos. También destaca Scratch, un lenguaje de programación visual que se puede usar para desarrollar juegos y que va enfocado principalmente a los más jóvenes. Por supuesto, también tenemos Python 2 y Python 3. De este lenguaje de programación, base de nuestro TFG, ya hablaremos de forma extendida más adelante.

Además de lo ya mencionado, en el menú tenemos los programas de LibreOffice para escribir, usar hojas de cálculo, presentaciones, etc. (la versión libre de Word, Excel, Powerpoint, etc.). También tenemos acceso directo al correo (Claws Mail) y a varias aplicaciones, como la calculadora, visor de pdf, editor de texto o visor de imágenes. Tenemos un apartado de juegos, donde encontramos Python Games, una serie de juegos simples creados con Python como el Tetris o el Cuatro en Raya, y Minecraft Pi, una versión para Raspberry Pi del famoso videojuego Minecraft. Por último, terminando así nuestro repaso por la interfaz gráfica de Raspbian, tenemos el botón “Shutdown”, para apagar la computadora y salir.

2.2.1.3 Instalación de Raspbian

El sistema operativo puede instalarse desde la página oficial de Raspberry Pi: https://www.raspberrypi.org/downloads/. Ahí, nos proponen dos opciones.

La primera es descargar el paquete NOOBS (*New Out Of the Box Software*, aunque la palabra *noob* se puede traducir también como “novato”), que es el recomendado para nuevos usuarios. NOOBS es un instalador de diversos sistemas operativos (no únicamente Raspbian) para la Raspberry Pi. Su instalación es tan simple como descargar el fichero comprimido desde el enlace indicado más arriba y descomprimirlo dentro de una tarjeta SD. También se pueden comprar tarjetas SD con el paquete NOOBS preinstalado. A continuación, se deberá introducir la Tarjeta SD en la Raspberry Pi, conectarla a la corriente y empezar a ponerla en marcha. Nos dará la opción de instalar varios sistemas operativos: Raspbian, Pidora, OpenELEC, OSMC, RISC OS y ArchLinux, dejando claro que el primero es el recomendado. Una vez seleccionado el sistema deseado, empezará a instalarse, y a continuación se puede empezar a configurar la placa.

La segunda opción es descargar directamente Raspbian desde el mismo link indicado más arriba. Descargaremos el fichero comprimido de Raspbian Jessie, la última versión del sistema

operativo, como ya hemos contado antes. Una vez hecho, se debe descomprimir, dándonos una imagen .iso, que se deberá montar con ayuda de algún programa como DAEMON Tools Lite. Una vez hecho, se tendrá que introducir la Tarjeta SD en la placa, como en la opción anterior. Al encenderla, se instalará el sistema operativo y podremos configurarla. Esta segunda opción es la que se ha utilizado en este trabajo.

2.2.2. Configuración de la Raspberry Pi

Una vez instalado Raspbian, se nos abrirá una ventana que nos permitirá configurar la Raspberry Pi por primera vez. En caso de querer volver a hacerlo más adelante, bastará con ejecutar el comando *sudo raspi-config* en el intérprete.

Este término "sudo" se justifica en que GNU/Linux es un sistema multiusuario. Al ser así, los usuarios "normales" ven restringidas ciertas funciones. Sin embargo, existe un llamado Superusuario, al que se suele dar el nombre "*root*", que puede acceder a todo el ordenador sin ningún tipo de restricción. Normalmente, no accedemos a la Raspberry Pi como Superusuario, así que si queremos ejecutar algún comando como tal, debemos usar *sudo* previamente. Si quisiéramos ejecutar un *shell* como Superusuario, podemos usar el comando *sudo su.* Sin embargo, esto no es recomendable puesto que al hacerlo ya no tenemos nada que nos proteja frente a posibles errores, dejando el sistema expuesto. No todos los usuarios pueden emplear el comando *sudo*. El usuario pi, con el que nosotros accedemos a la Raspberry, tiene esa posibilidad. Si quisiéramos permitírselo a otro usuario, deberíamos añadirlo usando *visudo*.

Volviendo a lo que comentábamos, al iniciar la Raspberry por primera vez o ejecutar raspi-config, veremos esta ventana (Fig. 21):

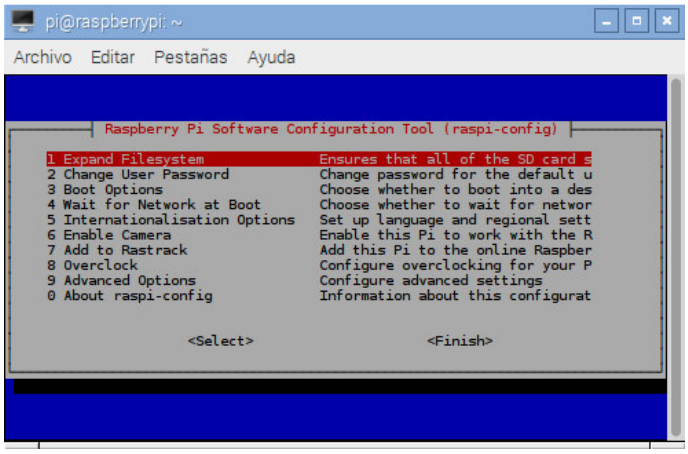


Fig. 21: Menú raspi-config

Procedemos a comentar brevemente todos los ajustes:

1. *Expand Filesystem* – Expandir el sistema de archivos: Cuando se ha instalado el sistema operativo, en la tarjeta SD la imagen sólo ocupa alrededor de 2 GB, cuando la capacidad de la tarjeta suele ser considerablemente mayor (16 GB en nuestro caso). Seleccionando esta opción, se puede ampliar el espacio en la tarjeta y aprovecharlo por completo.

2. *Change User Password* – Cambiar la contraseña del usuario: Por defecto, el usuario normal de la Raspberry es "pi" y la contraseña es "raspberry", ambos en minúsculas, además del usuario "root" del que ya hemos hablado. Con esta opción se puede cambiar la contraseña.
3. *Boot options* – Opciones de inicio: Se puede decidir si queremos ver la consola o el escritorio al iniciar la Raspberry. También se puede elegir si queremos que nos pida el usuario antes de ingresar, o queremos hacerlo directamente como "pi". Si iniciásemos en modo de comando y quisiéramos ingresar al modo gráfico, deberíamos ejecutar el comando *startx*.
4. *Wait for Network at Boot* – Esperar a la red al inicio: Sirve para decidir si se quiere esperar a que la Raspberry se conecte a la red al iniciarla antes de empezar a trabajar o si se desea acceder directamente aunque todavía no esté conectada.
5. *Internationalisation Options* – Opciones de Internacionalización: Permite cambiar el idioma del sistema operativo, la zona horaria y el teclado. Para elegir el español de España, se debe seleccionar es_ES.UTF-8. En cuanto a la zona horaria se ha elegido Europe/Madrid, y en cuanto al teclado se ha seleccionado "PC Genérico 105 teclas (intl) y a continuación "Español".
6. *Enable Camera* – Activar la cámara: esta opción activa el puerto para permitir la comunicación entre la CPU y el controlador de la cámara.
7. *Add to Rastrack* – Añadir a Rastrack: Rastrack es una página web; al activar esta opción se permite que la Raspberry Pi sea rastreada por esta página. Actúa simplemente como una herramienta estadística para contabilizar donde se encuentran las Raspberry Pi del mundo.
8. *Overclocking* – Overclocking: Se trata de una práctica para alcanzar una mayor velocidad de reloj en algún componente electrónico, para obtener un rendimiento más alto. Sin embargo, esto podría suponer una pérdida de estabilidad, una reducción de la vida útil del componente y una mayor generación de calor. Se ha decidido no aplicar *overclocking* en el presente trabajo.
9. *Advanced Options* – Opciones avanzadas: Aquí están las opciones avanzadas, que permiten hacer un *overscan* (borrar las líneas negras en algunos monitores), identificar la Raspberry Pi en la red local mediante un nombre, ver la distribución de la memoria, activar SSH (veremos esto más adelante) y actualizar el sistema.

2.2.3. Acceso a la Raspberry Pi mediante conexión remota

Se ha visto que la Raspberry Pi necesita unos cuantos elementos conectados a ella para poder utilizarla. A la hora de hacer pruebas, puede resultar algo engorroso tener que conectar la pantalla, teclado… cada vez que queremos utlizarla. Para evitar eso, se han estudiado métodos que permiten conectarse remotamente a la Raspberry Pi desde un portátil.

2.2.3.1 Averiguar la dirección IP de la Raspberry Pi

Un requisito indispensable para emplear la conexión remota es conocer la dirección IP asignada a nuestra Raspberry Pi. Para eso, se debe ejecutar el comando *sudo ifconfig* en el intérprete.

Al hacerlo, se obtiene el siguiente resultado, de donde se podrá extraer la dirección IP. Hay que mirar en el apartado eth0, ya que nos conectamos mediante Ethernet (Fig. 22):

```
pi@raspberrypi:~ $ sudo ifconfig
eth0      Link encap:Ethernet  HWaddr b8:27:eb:16:88:2c
          inet addr:192.168.1.12  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::b97a:db17:6a05:1ad/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1104 errors:0 dropped:12 overruns:0 frame:0
          TX packets:234 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:66978 (65.4 KiB)  TX bytes:41418 (40.4 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:496 errors:0 dropped:0 overruns:0 frame:0
          TX packets:496 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:40176 (39.2 KiB)  TX bytes:40176 (39.2 KiB)
```

Fig. 22: Dirección IP de la Raspberry Pi

Si queremos comprobar que la Raspberry ha quedado efectivamente bien conectada, se puede ejecutar comando *route –n* seguido de *ping 192.168.1.1* (Fig. 23). El primer comando permite ver hacia donde tenemos conexión, en el apartado Gateway. Con el segundo, se hace un ping a la dirección IP del router, lo que permite comprobar la conexión entre los dos dispositivos. Se verifica que se envían y reciben correctamente una serie de paquetes.

```
pi@raspberrypi:~ $ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.1.1     0.0.0.0         UG    202    0        0 eth0
192.168.1.0     0.0.0.0         255.255.255.0   U     202    0        0 eth0
pi@raspberrypi:~ $ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.73 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.758 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=0.818 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=64 time=0.710 ms
64 bytes from 192.168.1.1: icmp_seq=5 ttl=64 time=0.745 ms
64 bytes from 192.168.1.1: icmp_seq=6 ttl=64 time=0.802 ms
^C
--- 192.168.1.1 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5006ms
rtt min/avg/max/mdev = 0.710/0.927/1.734/0.364 ms
```

Fig. 23: Comandos para comprobar de la conexión

2.2.3.2 Conexión remota a la Raspberry Pi usando SSH

SSH, abreviatura de *Secure Shell*, en español “intérprete de órdenes seguro”, es un protocolo de comunicación que permite manejar un computador remotamente mediante un intérprete de comandos. Un protocolo de comunicaciones es un sistema de reglas necesario para dos entidades puedan establecer una comunicación entre ellas. Para llevar a cabo dicho acceso remoto, se debe tener acceso a la misma red IP donde está situada la Raspberry Pi desde nuestro ordenador. SSH usa técnicas de cifrado para que la información que viaja sea ilegible, evitando que nadie pueda descubrir el usuario y la contraseña de la sesión. Este cifrado supone una mejora respecto a otros protocolos como FTP o Telnet.

Para instalar el servicio SSH en la Raspberry Pi, se debe ejecutar ejecutar el comando *sudo apt-get install ssh*, aunque es posible que ya esté instalado predeterminadamente en las versiones más nuevas. A continuación, se debe iniciar el servicio con el comando *sudo /etc/init.d/ssh*

start. Por último, para que se ejecute automáticamente cada vez que encendamos la Raspberry Pi, se debe ingresar el comando: *sudo update-rc.d ssh defaults*

Una vez activado, desde el PC se puede descargar el programa Putty desde el siguiente enlace: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html eligiendo la opción "installer". Una vez instalado el programa, se ejecuta. En la ventana que se abrirá (Fig. 24), se debe escribir la dirección IP de la Raspberry Pi y pulsar sobre "Open". Se puede guardar la sesión dándole un nombre y pulsando sobre "Save". Al hacer eso, la siguiente vez que se desee acceder al Shell de la Raspberry Pi bastará con cargarlo pulsando "Load", ahorrándonos escribir cada vez la dirección IP.

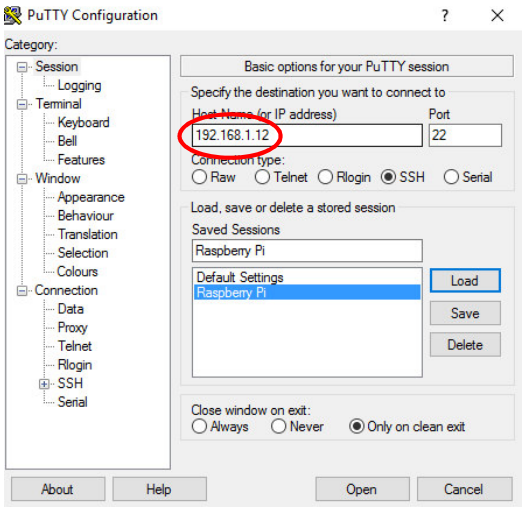


Fig. 24: Pantalla de inicio de Putty

Una vez iniciado, la aplicación pedirá el usuario y la contraseña. Cuando se ingresen ambos datos, se tendrá acceso al intérprete de comandos.

Queremos dejar claro que el uso de SSH únicamente nos permite acceder al intérprete de comandos de la Raspberry Pi, pero no al entorno gráfico. Si queremos tener acceso remoto al escritorio, debemos utilizar un servidor VNC.

2.2.3.3 Conexión remota a la Raspberry Pi usando VNC

VNC son las iniciales de *Virtual Network Computing*, que en español sería Computación Virtual de Red. Se trata de una aplicación de software libre cliente-servidor, que permite tomar el control remotamente de un ordenador servidor (en este caso, la Raspberry Pi) a través de un ordenador cliente (en este caso, el portátil), permitiéndo acceder al entorno gráfico de Raspbian. Una ventaja de este servicio es que funciona aun cuando el cliente y el servidor utilizan sistemas operativos diferentes, como en el presente caso caso.

Su funcionamiento consiste en que el servicio VNC instalado en la Raspberry Pi envía varias imágenes del escritorio cada segundo al cliente, permitiendo al usuario verlo y actuar sobre él remotamente.

En cuanto a la seguridad, por defecto VNC no es un protocolo seguro. El password se envía en un texto plano, por lo que se podría robar la contraseña si tanto ella como la clave de cifrado

son interceptadas desde una red. Sin embargo, se puede tunelar con otros protocolos como SSH o VPN que permitirían añadir un cifrado más seguro. Además, algunos de los programas empleados que permiten este acceso remoto, como el usado aquí, permiten cifrar la sesión.

El uso del protocolo VNC es muy interesante, y presenta aplicaciones interesantes, como en la enseñanza, donde el profesor podría compartir su pantalla con los alumnos, o en el mantenimiento de equipos, donde un técnico podría acceder remotamente al ordenador de un cliente inexperto para resolver su problema.

El primer paso para empezar a usar una conexión VNC es instalar el servidor VNC en la Raspberry Pi. Para eso, debemos ejecutar el comando *sudo apt-get install vncserver*. Una vez instalado, iniciaremos el servicio mediante el comando: *vncserver :1 –geometry 1280x800 –depth 16 –pixelformat rgb565*. ":1" es el número del escritorio que se está ejecutando y es requisito indispensable para acceder remotamente al entorno gráfico de la Raspberry Pi. Los otros datos son modificadores, para cambiar el tamaño de la pantalla ("geometry"), la profundidad del color ("depth") o la presentación del color ("pixelformat").

La primera vez que iniciemos el servicio, nos pedirá una contraseña para dar acceso al escritorio remoto. Si quisiéramos cambiarla, deberíamos ejecutar el comando *vncpasswd*.

Como programa para el acceso remoto por servidor VNC, se ha empleado VNC Viewer, que se puede descargar en este enlace: https://www.realvnc.com/download/viewer/. Una vez instalado, se ejecuta y se abrirá esta ventana (Fig. 25):

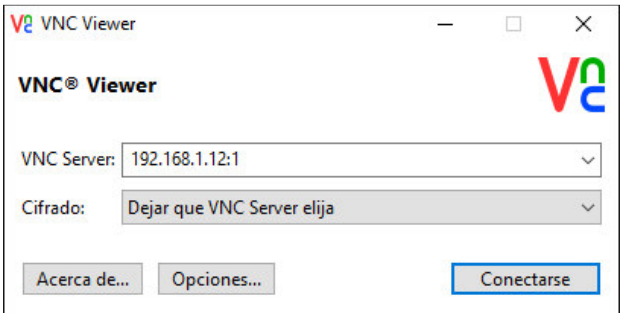


Fig. 25: VNC Viewer

Se deben introducir la dirección IP de la Raspberry Pi y el número de escritorio que elegido anteriormente. Como hemos comentado, este programa da la opción de cifrar la comunicación. Una vez esté todo, se pulsa sobre el botón "Conectarse". El programa pedirá la contraseña seleccionada previamente. Una vez introducida, se tendrá acceso al entorno gráfico de Raspbian remotamente.

2.2.4. Python

Hemos visto que en Raspbian tenemos disponibles varios lenguajes de programación, como algunos entornos de desarrollo de Java o el lenguaje Scratch, más enfocado para niños. Sin embargo, el más relevante y el que más interés pone la fundación Raspberry Pi en fomentar es Python.

2.2.4.1 ¿Qué es un lenguaje de programación?

Un lenguaje de programación es un conjunto de símbolos y reglas establecidas para unir dichos símbolos diseñados para crear procesos que puedan ser llevados a cabo por un ordenador, ya sea para crear programas que controlen un computador como para escribir algoritmos o simplemente comunicarse con otras máquinas. La programación es el proceso por el cual se crea un programa en la computadora. Este proceso de creación comprende el desarrollo del programa, la escritura del programa empleando un lenguaje específico, la compilación del programa, las pruebas y, finalmente, el desarrollo de la documentación. No hay que confundir un lenguaje de programación con un lenguaje informático. Este último concepto incluye tanto los lenguajes de programación como otros, por ejemplo los lenguajes para creación de páginas web, como HTML.

Los lenguajes de programación tienen varios elementos característicos: principalmente están las variables (por ejemplo “int”, “float” – para números, enteros o reales respectivamente –, “char”, “string” – caracteres o cadenas de caracteres – o “boolean”) y los vectores (conjuntos de variables). También existen los condicionales (“if”, “else”) o los bucles (“for”, “while”). Todos estos suelen ser comunes o muy similares en los distintos lenguajes.

A continuación, se puede hablar de las funciones, líneas de código recogidas bajo un nombre concreto, que se agrupan para volver a usarlas más adelante en el programa, ya sea una única vez o varias veces. En Python también existen los métodos, funciones asociadas a objetos de alguna clase, que ya se verán más adelante.

Todo lo mencionado en los párrafos anteriores, además de otros términos, puede incluirse dentro del grupo de sintaxis de un lenguaje. Esta consiste en la forma visible del lenguaje de programación, es decir, todas las sentencias de texto características de estos lenguajes que tienen una función concreta. Un ejemplo podría ser el comando “print” en Python o “System.out.println” en Java, que se usa para mostrar un texto por pantalla. Además de una sintaxis correcta, es decir, un uso adecuado de las sentencias, paréntesis…, también es necesario que lo que se escriba tenga sentido – que sea correcto y sea sintácticamente correcto.

A la hora de escribir un programa de la mejor manera posible es necesario tener unas cuantas cosas en cuenta. En primer lugar, es necesario que el programa sea correcto, es decir, que haga todo lo que se estableció que debía hacer antes de comenzar a desarrollarlo. Por eso, es importante tener claro, y a ser posible escribir, un pliego de condiciones para tener claro hacia dónde debe enfocarse el programa. En segundo lugar, el programa debe ser lo más claro y legible posible. Esto facilita tanto su desarrollo como su posterior comprobación y corrección, y además permite que en caso de que otro programador acabe trabajando con nuestro programa pueda leerlo e interpretarlo fácilmente. Relacionado con esto tenemos, en tercer lugar, la importancia de la eficiencia, que se puede explicar como que el proyecto, además de hacer lo que debe hacer (eficacia), lo haga utilizando la menor cantidad de recursos – memoria utilizada, espacio en disco, líneas de código – posible (eficiencia). Por último es importante la portabilidad del programa, que pueda ejecutarse en Hardware y Software distintos. Este punto es especialmente relevante en este proyecto porque se ha desarrollado inicialmente el programa en un PC que usa Windows, y luego se ha pasado a la Raspberry Pi que usa Raspbian.

Ejemplos de lenguajes de programación conocidos son Java, C++ o Python.

2.2.4.2 Breve historia e información de interés

Python (Fig. 26) es un lenguaje de programación creado en los años 80 por el holandés Guido van Rossum cuando trabajaba en el CWI (Centrum Wiskunde & Informatica), un centro de investigación de los países bajos. Fue creado como sucesor del lenguaje de programación ABC, en cuyo desarrollo participaba el propio van Rossum. El nombre Python viene del grupo de humoristas británicos Monty Python, del que es seguidor su creador.

Fig. 26: Logo de Python

La implementación del lenguaje empezó en 1989, y dos años más tarde empezaron a salir las versiones 0.9. En 1994 surgió la versión 1.0, 6 años más tarde la versión 2.0 y en 2008, la 3.0. La versión más reciente es la 3.5, lanzada en septiembre de 2015, aunque también se sigue usando la versión 2.7, pese a ser más "antigua" (aunque es posterior a la versión 3.0, ya que fue lanzada en 2010). Más adelante, hablaremos de las diferencias entre las distintas versiones.

Tanto la versión 2 como la versión 3 se pueden descargar en la página oficial de Python, si no estuviera disponible por defecto en nuestro sistema operativo (en la Raspberry Pi ya viene preinstalado): https://www.python.org/. Además de descargar el programa, en la página tenemos muchos más contenidos, como documentación y bibliografía para programadores y desarrolladores, noticias, historias de hechos relacionados con Python, etc.

En España, tenemos la Asociación Python España, creada en 2013 y que se define como "una asociación sin ánimo de lucro cuyo propósito es promover el uso del lenguaje de programación Python en España, servir como punto de encuentro a aquellos interesados en su uso y darles soporte en la medida de sus posibilidades". Esta asociación organiza una conferencia anual en nuestro país llamada PyConES y ayuda económicamente a las comunidades locales que organizan eventos relacionados con Python. A cambio, sus socios pueden obtener descuentos en libros, participar en sorteos… entre otras ventajas. Más información en su página web: http://www.es.python.org/.

Python forma parte desde 2003 del top ten de los lenguajes de programación más populares, según el índice TIOBE. A principios del presente año 2016 estaba situado en la quinta posición. Python es usado por organizaciones importantes como Google, Yahoo! o la NASA, que explotan comercialmente este lenguaje. Python ha sido y es utilizado para aplicaciones web, videojuegos, frameworks gráficos, sistemas embebidos y muchas otras aplicaciones informáticas, como Dropbox y Torrent. Además, ha servido de base para el desarrollo de otros lenguajes derivados, como puede ser Ruby.

2.2.4.3 Características de Python

Python sigue una filosofía similar a la de Unix, llamada Filosofía Python, según la cual el código debe seguir los principios de legibilidad y transparencia, con el fin de fomentar un lenguaje que sea lo más simple posible, sintácticamente hablado. Otra base que emplean los desarrolladores de Python es que el lenguaje sea divertido de usar. Así, tenemos que al código que sigue esta filosofía se le llama código "pythonico", mientras que del que no cumple esta filosofía se dice que es "no pythonico". Los principios de esta filosofía son los siguientes:

- "Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Disperso es mejor que denso.
- La legibilidad cuenta.
- Los casos especiales no son tan especiales como para quebrantar las reglas.
- Lo práctico gana a lo puro.
- Los errores nunca deberían dejarse pasar silenciosamente.
- A menos que hayan sido silenciados explícitamente.
- Frente a la ambigüedad, rechaza la tentación de adivinar.
- Debería haber una -y preferiblemente sólo una- manera obvia de hacerlo.
- Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés.
- Ahora es mejor que nunca.
- Aunque *nunca* es a menudo mejor que *ya mismo*.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede que sea una buena idea.
- Los espacios de nombres (*namespaces*) son una gran idea ¡Hagamos más de esas cosas!"

Principios recogidos en: Tim Peters (2004). The Zen of Python.

Python es conocido como un lenguaje programación de alto nivel. Un lenguaje de alto nivel se caracteriza por que su código es expresado de una forma similar al lenguaje humano, difiriendo considerablemente del lenguaje empleado por las máquinas. Los lenguajes cercanos al lenguaje máquina, que se orientan a áreas específicas, son conocidos como lenguajes de bajo nivel. La aparición de los lenguajes de programación de alto nivel a la necesidad de que los usuarios comunes pudieran solucionar el procesamiento de datos de forma más sencilla. El uso del término "alto nivel" se debe a se considera que estos lenguajes están en el nivel más alto de abstracción del lenguaje de máquina, puesto que en vez de tratar con registros o direcciones de memoria se trata con variables, objetos o funciones, elementos totalmente abstractos en informática. Esto proporciona ventajas interesantes como una mayor sencillez, la validez del código en distintos hardware o software y la posibilidad de crear programas complejos en pocas líneas. A cambio, se cede perdiendo algo de velocidad.

Se dice que Python es un lenguaje de propósito general, un lenguaje que se puede usar para diferentes propósitos como acceso a bases de datos, comunicación entre dispositivos, cálculos matemáticos, etc.

Python es, tradicionalmente, un lenguaje de programación interpretado. Un lenguaje interpretado es aquel que, al contrario que en los lenguajes compilados, no requiere que su código fuente se pase por completo a lenguaje máquina antes de ejecutarlo, proceso conocido como compilación. Para hacer eso, se necesita lo que se conoce como un intérprete, un programa que leerá cada sentencia de nuestro código una a una, cuando necesite ser ejecutado, y la traducirá a lenguaje de máquina, sin guardar el resultado de esta traducción. La ventaja principal de usar un intérprete es que es independiente de la máquina, por lo que un programa escrito en cierto lenguaje podrá ser ejecutado en cualquier máquina que tenga un intérprete para dicho lenguaje. Además, un lenguaje interpretado permite hacer pequeñas modificaciones sin vernos obligados a compilarlo todo de nuevo. A cambio, el programa interpretado será más lento que el programa compilado, ya que cada sentencia se traduce una a una. Tampoco se podrá garantizar la seguridad del código fuente, cosa que sí se puede hacer en un lenguaje de programación compilado. Un ejemplo de lenguaje compilado es C++, y un ejemplo de lenguaje interpretado es el lenguaje de MATLAB.

Sin embargo, en la actualidad se usa mucho lo que se llama lenguajes de programación intermediarios. En este tipo de lenguajes no se transforma el código en lenguaje máquina como en los compilados, sino que se pasa a un lenguaje intermedio para, a continuación, compilar las sentencias una a una, en tiempo de ejecución, por lo que también se le conoce con el nombre de programación en tiempo de ejecución. Esta técnica permite aprovechar las ventajas tanto de los lenguajes compilados como de los interpretados, ya que garantiza la portabilidad y presenta un tiempo de ejecución menor que los interpretados, aun sin llegar a la velocidad de los lenguajes compilados. Este es el acercamiento que presentan algunos lenguajes, como Java o el lenguaje que se está estudiando: Python.

Antes de continuar enumerando las características de Python, vamos a hablar de un concepto importante a la hora de entender correctamente el funcionamiento de un lenguaje de programación: el paradigma de programación. Un paradigma de programación es una propuesta hecha por desarrolladores, un enfoque particular que deben funcionar como base para diseñar soluciones. Así, a la hora de resolver un problema concreto siempre se utilizará el mismo enfoque y la misma forma de abstraer los elementos del problema para alcanzar una solución aceptable. Los paradigmas de programación se pueden separar en dos tipos principales: programación imperativa y programación declarativa, que a su vez han ido dando lugar a subcategorías que se han acabado convirtiendo también en nuevos paradigmas. La programación imperativa ha dado lugar a la programación orientada a objetos – la más usada actualmente – y a la programación dinámica. De la programación declarativa nacen la programación funcional, la programación lógica y la programación con restricciones.

Python es un lenguaje multiparadigma. Eso significa que puede soportar más de un paradigma de programación. Concretamente, acepta la programación imperativa, la programación orientada a objetos y la programación funcional.

La programación imperativa, también llamada programación por procedimientos, se caracteriza en dar instrucciones al ordenador, en forma de algoritmos, sobre qué cosas hacer y cómo hacerlas. Este paradigma es el más usado, y también el más antiguo, ya que el propio Lenguaje de máquina emplea la programación imperativa, por lo que prácticamente todo el hardware de las computadoras utiliza el paradigma de programación imperativa. Otros ejemplos de lenguajes puramente imperativos son Fortran, C y BASIC.

La programación orientada a objetos (abreviada como POO), el paradigma más usado en la actualidad, surgió en los años 80 y se ha ido asentando desde entonces. Deriva de la programación imperativa, en cuanto a que también consiste en dar una serie de instrucciones para resolver un problema. Sin embargo, tiene una peculiaridad esencial con respecto a esta, que es el uso de unos elementos llamados objetos, que son la base de todo el código, y que forman parte de una clase. Una clase es una construcción sintáctica que pretende englobar elementos con características y propiedades comunes, llamadas atributos, que pueden ser propias o heredadas de otra clase. Un objeto es una instancia de una clase, esos elementos que forman parte de las clases. Cada objeto tiene identidad – para distinguirlos de otros objetos–, comportamiento – pueden realizar tareas – y estado. Los objetos pueden tener asociados métodos, que son algoritmos asociados a un objeto y que se ejecutan tras recibir un mensaje, una petición dirigida a un objeto.

La popularidad del uso de la POO se debe principalmente a que la forma de resolver los problemas usando este paradigma se acerca a la forma de pensar de los seres humanos y a nuestra forma de actuar en otros dominios. Además, se puede incrementar la productividad y la fiabilidad de los programas, gracias a aplicaciones como la reutilización de componentes, mediante composición o herencia. También, este paradigma permite una transición fácil y cómoda desde otros lenguajes de programación. Todas estas y otras ventajas hacen que varios de los lenguajes de programación más usados, como C++, Java o Python, usen la POO.

El tercer y último paradigma soportado por Python es la programación funcional. Se trata de una variante de la programación declarativa, que es considerada la opuesta de la imperativa, un paradigma basado en la declaración o especificación de condiciones, proposiciones, afirmaciones, ecuaciones, transformaciones y restricciones para describir el problema y así llegar a la solución del problema. En un programa hecho siguiendo la programación imperativa, se describen los pasos necesarios para llegar a la solución del problema. En cambio, en un programa hecho con un paradigma de programación declarativa, se describe el problema, y se resolverá mediante mecanismos internos. Este paradigma tiene una gran base matemática, y por eso sirve para desarrollar lenguajes lógicos y algebraicos como Prolog o SQL. De este paradigma desciende la programación funcional, que está basada, como su propio nombre indica, en el uso de funciones matemáticas. Ejemplos de lenguajes que emplean la programación funcional son Haskell y Lips.

Python emplea el tipado dinámico. Esto significa que la comprobación del tipo de una variable – int, float, string, etc. – se realiza durante la ejecución. El tipado dinámico es característico de los lenguajes interpretados y permite que una variable pueda tomar valores de distinto tipo, aumentando la flexibilidad. Este tipado es opuesto al tipado estático, en el cual la comprobación

del tipo de variable se hace durante la compilación, y no durante la ejecución, obligando a que una variable deba tener un tipo bien definido, aunque esto permite que los posibles errores de programación sean detectados antes. El tipado dinámico se ve muy claramente en Python, porque a la hora de usar una variable no hay que declarar el tipo de variable previamente, algo que no ocurre en lenguajes estáticos como C++ o Java.

Para gestionar la memoria de forma eficiente Python emplea dos técnicas: el conteo de referencias, que permite contar cuántas veces el programa está refiriendo a un recurso y liberar la memoria que le corresponde si no está siendo utilizado; y el recolector de basura, que puede liberar los recursos que se están utilizando de forma cíclica pero que no tengan utilidad.

Python es multiplataforma, siguiendo así una línea coherente con los sistemas operativos basados en Unix de los que ya hemos hablado largo y tendido. Esto hace que este lenguaje sea compatible con distintos hardware y distintos software.

Python tiene una licencia de código abierto conocida como Python Software Foundation License, que garantiza la libertad de los usuarios de usar, estudiar y modificar libremente el código fuente, sin que sea necesario que los trabajos derivados sean también código abierto. Esta licencia es compatible con la licencia pública general de GNU, haciéndolo totalmente aceptado por el movimiento del software libre.

2.2.4.4 Implementaciones y entornos de desarrollo integrados

Cuando hablamos de un lenguaje de programación, una implementación se refiere a un sistema en el cual se ejecutarán los programas que realicemos. La implementación generalmente será un intérprete o el conjunto de un compilador con un intérprete, dependiendo de si el lenguaje es de tipo interpretado o compilado. Generalmente, en el caso de Python, cuando hablemos de implementación nos estaremos refiriendo al intérprete. La implementación principal para Python es la llamada CPython, un intérprete escrito en C, soportado en diversas plataformas. Otros intérpretes conocidos son PyPy, RPython, Jython o IronPython, escrito en lenguaje C#.

Un entorno de desarrollo integrado (IDE, del inglés *Integrated Development Environment*) es un programa informático cuya finalidad es proporcionarle facilidades al programador o desarrollador en el desarrollo de los programas. Los IDE suelen tener un editor de código fuente, un depurador, un intérprete o un compilador (o ambos) y un auto-completado inteligente de código. También podemos encontrar IDE con herramientas que facilitan el desarrollo de interfaces gráficas de usuario.

La mayor parte de las implementaciones de Python funcionan como intérpretes de comandos en los que se reciben los comandos secuencialmente y se muestra el resultado inmediatamente, lo que se conoce como REPL (*read-eval-print loop*). Básicamente, Python actúa como una interfaz de línea de comandos o *shell*. Sin embargo, los IDE pueden añadir funcionalidades distintas a las de intérprete básico, permitiéndonos por ejemplo redactar scripts para que sean ejecutados más adelante. Hay diversos IDE que se pueden utilizar en Python, pero el más importante, y que además viene preinstalado en la Raspberry Pi, es IDLE.

Las siglas IDLE vienen de *Integrated DevoLopement Environment*. Es un entorno de desarrollo creado con la intención de mostrar un funcionamiento simple y ayudar a los novatos a iniciarse,

sobre todo en el ámbito educacional. IDLE está codificado al 100% en lenguaje Python, y la parte gráfica se ha realizado utilizando la librería “tkinter”, de la que ya hablaremos más adelante, de forma bastante sobria pero eficaz. Es multiplataforma, por lo que funciona de forma similar en los diferentes sistemas operativos.

La ventana principal de IDLE (Fig. 27) es un intérprete interactivo, equivalente al shell de Python. Tiene color para sentencias de Python y palabras reservadas, muestra salidas y señala errores. También existe la opción de abrir el editor de texto, desde el que se podrán escribir scripts. El editor de texto presenta, entre otras características, coloreado de código, ayuda sobre los métodos o funciones y auto-completado inteligente. IDLE tiene un buen buscador, que permite realizar la búsqueda tanto en la ventana principal como en el editor. Aparte de esto, también se pueden ver funciones más características, como la configuración o el menú de ayuda. En la ventana del editor hay opciones como la de identar o quitar la identación (ver apartado 2.2.4.5 para saber qué es la identación) a un conjunto de líneas o insertar o quitar comentarios, entre otras cosas. También se puede verificar la sintaxis del código y ejecutarlo.

Una segunda opción es utilizar un editor de texto en el que escribir el programa. El recomendado, utilizado en este proyecto, es Sublime Text. Se puede descargar desde http://www.sublimetext.com/. Una vez instalado, para configurarlo y poder trabajar con Python, se deben instalar unos paquetes. Lo primero que hay que hacer es entrar en https://packagecontrol.io/, pulsar sobre “Install Now” y copiar el texto de la Fig. 28.

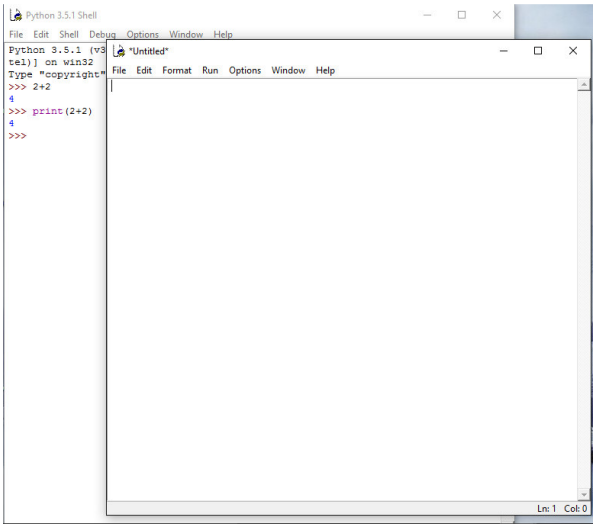


Fig. 27: IDLE de Python

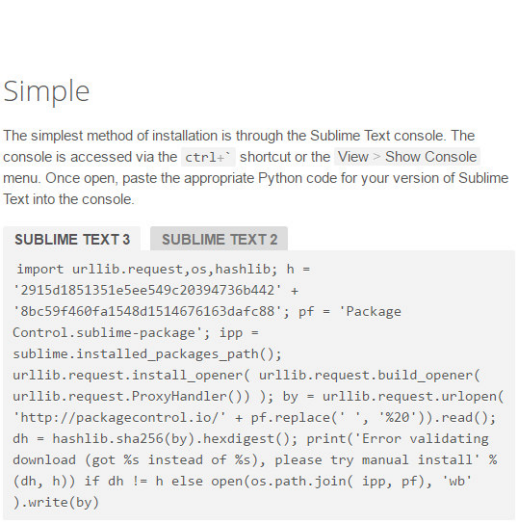


Fig. 28: Package control

Volviendo a la interfaz de Sublime Text, se debe pulsar sobre View -> Show Console. Se abrirá una pequeña consola desde la que hay que pegar el texto copiado anteriormente. Una vez hecho esto, pulsamos Ctrl + Shift + P y se abrirá una ventana desde la que se debe buscar el Package Control e instalarlo (Fig. 29):

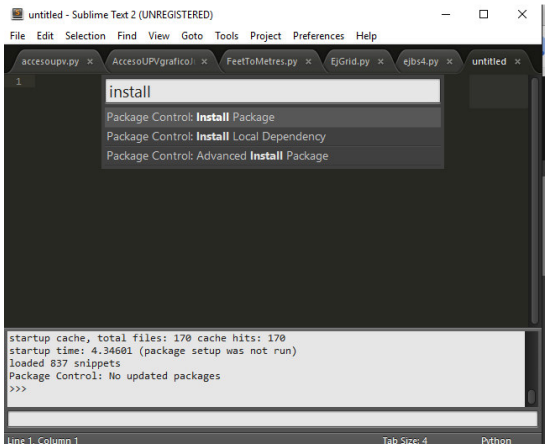


Fig. 29: Consola e Instalación del Package Control

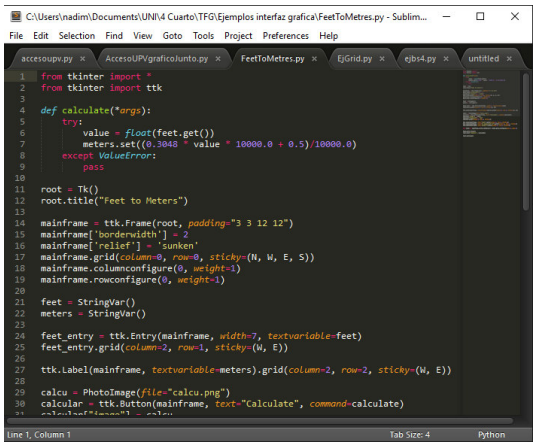


Fig. 30: Editor de texto de Python (Sublime Text)

Al hacerlo, se abrirá un recuadro, desde el que hay que buscar SubimeREPL e instalarlo. Una vez hecho eso, cuando se quiera usar el *shell* de Python desde Sublime Text bastará con pulsar sobre Tools -> SublimeREPL y seleccionar Python. Si se pulsa en la esquina inferior derecha, donde debería poner "Plain Text" al abrir el programa, y se selecciona Python, se podrá utilizar el editor de texto de Python.

Se abrirá una interfaz similar a la del IDLE de Python (Fig. 30), desde la cual se puede escribir el código de forma agradable y cómoda:

2.2.4.5 Elementos del lenguaje Python

Como ya se ha mencionado anteriormente, Python es un lenguaje de programación ideado para ser lo más simple posible y poder ser leído con facilidad. A continuación, se enumerarán sus elementos y se explicarán las principales características de su lenguaje.

Los tipos de datos principales son los siguientes (Tabla extraída de Wikipedia):

Tipo	Clase	Mutable o inmutable	Ejemplo
str	Cadena	Inmutable	"Hola Mundo"
unicode	Cadena	Inmutable	u"HolaMundo"
list	Secuencia	Mutable	[4.0, "Hola", True]
tuple	Secuencia	Inmutable	(4.0, "Hola", True)
set	Conjunto	Mutable	set([4.0, "Hola", True])
frozenset	Conjunto	Inmutable	frozenset([4.0, "Hola", True])
bytearray	Secuencia	Mutable	bytearray([115, 46, 78, 119])
bytes	Secuencia	Inmutable	bytes([115, 46, 78, 119])
dict	Mapping	Mutable	{'key1': 4.0, 3: true}
int	Número entero	Inmutable	12
long	Número entero	Inmutable	52154124775487889L

float	Número decimal	Inmutable	2.895
complex	Número complejo	Inmutable	3 + 4j
bool	Booleano	Inmutable	True o False

Tabla 2: Tipos de datos en Python

La mutabilidad se refiere a si el contenido o el valor del dato puede variarse en tiempo de ejecución – mutable – o no – inmutable.

Algunos de estos tipos de datos, como “int”, “float”, “str” o “bool”, son conocidos de sobra, pero otros merecen un par de líneas para dejar claro en qué consisten.

Las listas y las tuplas son conjuntos de datos del mismo o de distinto tipo, aunque las listas se suelen usar para elementos del mismo tipo en cantidad variable y las tuplas para elementos de distinto tipo en cantidad fija, a pesar de que en la práctica se pueden usar como queramos. Básicamente, son similares a lo que son los vectores en otros lenguajes de programación. La principal diferencia entre ambos es la mutabilidad. Para acceder a un elemento de una lista o de una tupla se usa un número entero que actúa como índice, empezando por 0.

Los diccionarios (“dict”) son similares a las listas y a las tuplas, sólo que no se accede a sus elementos mediante un índice, sino que cada elemento se ha asignado a una clave, que puede ser cualquier tipo de dato inmutable. Al no haber índice, tampoco hay un orden definido.

Los conjuntos se escriben como set(ítems) o frozenset(ítems), donde ítems debe ser un dato iterable como una lista o una tupla. Los conjuntos no mantienen el orden, y además no pueden contener elementos repetidos. Se usan principalmente para eliminar duplicados o para hacer operaciones matemáticas de unión, intersección, etc.

Conviene recordar que las variables, que pueden ser de cualquiera de los tipos mencionados en la tabla anterior, se definen dinámicamente. Esto significa que no es necesario indicar previamente el tipo de la variable, e implica que una misma variable puede presentar dos tipos distintos a lo largo de nuestro programa. Todas están variables son objetos, ya que Python emplea la Programación Orientada a Objetos, y descienden de una clase. Como objetos que son, cada uno tiene sus propios métodos asociados.

Python, como el resto de los lenguajes de programación, tiene una serie de palabras claves, o palabras reservadas, que tienen una función concreta y no pueden ser usadas como identificadores. Estas son:

and	def	for	None	return
as	del	from	nonlocal	True
assert	elif	global	not	try
async	else	ir	or	while
await	except	import	pass	with
break	exec	in	print	yield
class	False	is	raise	
continue	finally	lambda	range	

Tabla 3: Palabras reservadas de Python

Algunas de estas palabras evidencian lo que se comentaba antes acerca de que Python aspira a ser un lenguaje que pueda ser leído con facilidad. Algunos símbolos habituales en programación como "!", "||" o "&&" se han sustituido por "not", "or" y "and", respectivamente.

Otro ejemplo de esta mencionada simpleza es que Python no sigue la línea de la gran mayoría de lenguajes de programación de agrupar un bloque de códigos entre corchetes "{}". A cambio, los distintos bloques se separan mediante tabulación, lo que se conoce como identación obligatoria. Esta mayor simpleza y comodidad que nos proporciona la identación viene a cambio de una pérdida de robustez, ya que es relativamente fácil cometer un error con la identación que haría que el código deje de ser correcto.

Como en otros lenguajes de programación, en Python se pueden introducir comentarios, líneas que en general tienen una función explicativa que el intérprete no tiene en cuenta. Existen dos maneras para hacerlo: la primera es escribiendo el comentario entre tres comillas, como por ejemplo '''comentario'''; la segunda es anteponiendo el símbolo "#", extendiendo el comentario hasta el final de la línea. El primer modo permite insertar comentarios de varias líneas, mientras que el segundo sólo permite añadir comentarios de una línea.

En Python, las funciones se definen usando la palabra reservada "def", seguida del nombre de la función y, entre paréntesis, los parámetros que recibe si se diera el caso. Si se quiere que la función devuelva un valor – cuyo tipo no sería necesario indicar, por supuesto – se debe indicarlo mediante la instrucción "return".

Como lenguaje orientado a objetos, Python también presenta clases – que, técnicamente, también son objetos, instancias de una "metaclase" –, que deben ser declaradas usando "class", seguido del nombre de la clase. En el interior de una clase se pueden declarar funciones, que aquí se llamarán métodos, y que podrán ser referenciados desde fuera.

Los condicionales se introducen usando "if" – primera condición –, "elif" – siguientes condiciones, hasta la penúltima – y "else" – última condición –, con las instrucciones que se incluyen dentro de cada una correctamente indexadas. Cada condición es evaluada secuencialmente hasta encontrar la primera que sea verdadera, o llegar al bloque else y ejecutar su instrucción.

El bucle "for" funciona de una forma distinta a lenguajes como C++ o Java. En Python, se introduce usando la palabra "for" seguida de un nombre de variable, seguido de "in" y seguido de un iterable. El iterable puede ser una lista, una tupla o una cadena, y por cada elemento que este iterable presente se ejecutará el código contenido dentro del bucle, además de añadirle a la variable especificada el contenido del elemento del iterable. En cuanto al bucle "while", este sí que funciona de la misma forma que se suele a ver en otros lenguajes de programación: evalúa una condición, si es verdadera ejecuta el código interno al bucle, volverá a evaluar… y así sucesivamente mientras la condición sea verdadera.

La librería estándar de Python es muy reputada y se considera una de sus grandes ventajas, ya que nos provee herramientas para realizar diversas tareas. Sin embargo, en caso de que se quisiera expandir esta funcionalidad, está permitido importar módulos o librerías gracias al comando "import" seguido del nombre del módulo o librería. Los módulos son un conjunto de

funciones agregables a Python para realizar ciertas funciones, como el módulo "os" que permite acceder a ciertas funciones del sistema operativo; de las librerías hablaremos en el apartado 3. A principios del año 2016 había 72000 paquetes muy diversos disponibles para instalar en Python. En cuanto a las librerías, se hablará de ellas en el siguiente apartado.

Otro comando que merece la pena mencionar es el "try", que como su propio nombre indica "intenta" ejecutar la instrucción que le viene a continuación, pero en caso de no poder hacerlo admite excepciones mediante el comando "except". Esto se usa generalmente para proteger nuestro código frente a errores. Además de estos dos comandos, la secuencia puede cerrarse con el comando "finally", que garantiza que las instrucciones que tiene a continuación se ejecutarán siempre, sin importar las excepciones anteriormente mencionadas.

Hay muchos otros comandos como "with", "assert" o "yield", pero no se han explicado ya que son menos habituales y no han sido empleados en este programa.

En cuanto a las operaciones matemáticas, Python acepta los operadores aritméticos de C y los lenguajes basados en C: "+", "-", "*", "/" y "%". También presenta el operador "**" para exponenciales y, actualmente, un operador para matrices "@". Tendremos disponibles muchos otros métodos si importamos el módulo "math", como por ejemplo para hacer raíces cuadradas – math.sqrt() – o logaritmos – math.log().

En cuanto a las comparaciones, mencionar que "==" compara el valor de dos variables y devuelve "True" si son iguales, mientras que "is" compara si dos variables señalan al mismo objeto, devolviendo "True" si fuera así. Python también tolera el enlazamiento de comparadores, como por ejemplo "a < b < c", donde se verifica que "a" es menor que "b" y que a su vez "b" es menor que "c", algo que en otros lenguajes no se puede hacer.

2.2.4.6 Diferentes versiones de Python

Como ya se ha contado anteriormente, en 2008 surgieron las versiones 3.X, llegando hasta la versión 3.5 en la actualidad, con la intención de dejar atrás las versiones 2.X. Aun así, en 2010 surgió la versión 2.7, a pesar de que ya estaba en el mercado la número 3. Aunque muchos desarrolladores se han pasado a la versión 3.5, algunos otros se mantienen "fieles" a la 2.7. En este apartado vamos a evaluar algunas diferencias entre ambas:

- A partir de la versión 3.0, la sentencia "print" se ha reemplazado por la función "print()". A efectos prácticos esto implica que, donde antes escribíamos *print "Hola mundo"*, ahora deberíamos escribir *print ("Hola Mundo")*. Lo mismo pasa con las ejecuciones que ahora también son una función "exec()".
- El operador de matrices "@" se incluyó a partir de la versión 3.5.
- A partir de la versión 3.0, la división *x/y* actúa como división real, por ejemplo: *5/2 == 2.5* y *6/2 == 3.0*. En cambio, la siguiente división *x//y* dividirá y, a continuación, truncará, por ejemplo: *5/2 == 2* y *6/2 == 3*.
- Desaparece el tipo de dato "unicode". Los datos string ya lo incluyen.
- Desaparece el tipo de dato "long", ahora se puede hacer todo con "int".

- En Python 2.X se recomienda que a la hora de crear una clase esta herede de “object”. A partir de Python 3.0 esto ya no es necesario.
- En las excepciones, cuando queramos pasar algún argumento, necesitamos introducirlo entre paréntesis.
- Cambios en el nombre de algunos atributos y métodos.
- Cambio en el nombre de algunos módulos o librerías, por ejemplo, la librería “http.cookiejar”, que hemos usado en el proyecto, anteriormente se debía importar como “cookielib”. También se han agrupado dos o más paquetes de módulos en uno solo.
- Para la entrada de datos sólo se utiliza la función “input()”, uniendo así las funciones “input()” y “raw_input()” que había en la versión 2.
- En las primeras versiones 2.X el método “format()”, para introducir datos por teclado, no existía.
- Antes de la versión 3.0 había dos tipos de clases: “old-style” y “new-style”, pero a partir de dicha versión todas las clases son “new-style”.

En este TFG se ha decidido emplear Python 3.5, porque se ha considerado que no hay nada que podamos hacer en Python 2.7 que no se pueda hacer a su vez en Python 3.5, y viendo eso la decisión tomada ha sido aprender a usar la versión más reciente.

Se podría contar mucho más sobre Python de lo que se ha hecho en las últimas páginas. La información compartida se ciñe a lo que se considera relevante sobre este lenguaje de programación y otras cosas interesantes para el desarrollo del proyecto. Aun así, cuando se trate el desarrollo del proyecto también se hablará de forma más extendida de las librerías de Python utilizadas, y posiblemente de otros aspectos secundarios de Python que puede ser interesante mencionar.

Con esto, damos por finalizado la primera parte de la memoria, en la que se ha explicado todo lo importante sobre el Hardware y el Software utilizado, y se puede pasar a hablar del desarrollo del proyecto.

3. Desarrollo del proyecto en Python

En este apartado se explica con todo detalle el desarrollo del proyecto. Se divide en dos grandes apartados: en primer lugar se hablará del desarrollo de la interfaz gráfica de nuestra aplicación y en segundo lugar, de toda la funcionalidad del terminal. A su vez, cada apartado se dividirá en dos partes: primero se comentarán las librerías utilizadas y después se tratará la programación.

También cabe comentar que en este apartado no se explica explica detalladamente cómo se debe usar la aplicación. Para eso, se les remite al apartado 4 del presente TFG, donde se podrá encontrar una breve guía de instrucciones.

3.1. Interfaz gráfica

3.1.1. Librerías utilizadas

Antes de empezar, conviene explicar brevemente el concepto de librería. Se trata de un conjunto de funcionalidades, escritas en algún lenguaje de programación, que se invocan para ser usadas atendiendo a una finalidad concreta. Una librería se diferencia de un programa ejecutable en que este último se utiliza de manera autónoma, mientras que la librería está pensada para ser usada por los propios programas de forma independiente. Las librerías suelen tener objetos, métodos y atributos que permiten resolver problemas.

Este término, "librería", es el más utilizado, porque es similar al término utilizado en inglés, *library*, aunque la traducción más exacta sería "biblioteca". A efectos prácticos, los dos términos están aceptados, así que este texto se referirá a cualquiera de los dos términos indistintamente.

3.1.1.1. Tkinter

La librería gráfica por excelencia utilizada en el desarrollo de este TFG es Tkinter. Se trata de la librería estándar de Python para desarrollo de interfaces gráficas de usuario (GUI, *Graphical User Interface*) y está incluida en la versión Windows de Python.

Técnicamente, Tkinter es un binding de la biblioteca Tk del lenguaje Tcl, adaptado para Python. Un binding es una adaptación de una librería para que pueda ser utilizada en un lenguaje de programación distinto a aquel para el cual ha sido escrita. Así pues, Tk (cuyo nombre viene del inglés *tool kit*) es una librería creada para Tcl pensando en el desarrollo de GUI, y que se ha adaptado a otros lenguajes como Python, Perl y Ruby. El nombre Tkinter viene precisamente de *Tk Interface*.

La última versión disponible es Tk 8.5. Esta versión trae la novedad de la nueva funcionalidad ttk, que implementa cambios como una mejor integración de los widgets en el sistema operativo en el que ejecutamos el programa – algo que era la fuente de muchas críticas a Tk –, introducción de algunos nuevos widgets, nuevas fuentes, etc.

Hay bastante controversia en lo que respecta a Tkinter. Algunos critican duramente esta librería gráfica, diciendo que está algo limitada para realizar grandes proyectos, y atacan a su estética, principalmente en las versiones más antiguas. Otros, en cambio, defienden que estando bien

documentados sobre esta librería se pueden desarrollar interfaces gráficas de gran calidad, y que las dificultades que existían han sido solventadas de forma más que aceptable.

Se podrían haber elegido otras librerías gráficas, puesto que el abanico de opciones para Python es bastante grande: WxPython, PyQt, PyGTK, EasyGUI, etc. Pero después de valorar ventajas e inconvenientes de cada una – por ejemplo, PyQt requiere adquirir una licencia, otras presentan algún problema de integración y son poco "Pythonicas", etc. – se ha decidido quedarnos con Tkinter, principalmente porque es la más recomendable para aprendices, tiene más documentación disponible y, en caso de aprender a usarla correctamente, permite desarrollar interfaces gráficas de buena calidad.

3.1.1.2. PIL

PIL son las iniciales de *Python Imaging Library*, una librería de Python disponible para múltiples plataformas. PIL se utiliza fundamentalmente para manipular imágenes. Algunos de sus usos potenciales son cambio de tamaño, trabajo con transparencias, filtrado de imágenes (poner la imagen borrosa, con contorno…), retoque de imágenes (brillo, contraste, color…), añadir texto a las imágenes, etc. Básicamente, son funciones que se puede realizar desde cualquier programa de edición gráfica como Photoshop, pero de las que puede resultar interesante disponer en Python. PIL acepta múltiples formatos de imagen, como JPEG, PNG, GIF…

Concretamente, se ha utilizado esta librería para ajustar el tamaño de unas cuantas imágenes usadas en la interfaz. Se ha juzgado que sería más cómodo hacerlo directamente desde el propio código antes que recurrir a un programa de edición.

3.1.2. Código

Aquí, se explicará todo el código referido a la implementación de la interfaz gráfica, igual que en el apartado 3.2.2 se hablará de la implementación de la funcionalidad de la aplicación. Señalar que, aunque se irá explicando paso a paso todo lo que hemos hecho, en caso de querer consultar el código del programa al completo se puede encontrar en el anexo.

A la hora de empezar a implementar la interfaz gráfica de la aplicación, el primer paso es importar las bibliotecas necesarias, en este caso tkinter y sus librerías "derivadas" de interés (ttk para implementar los widgets, font para las fuentes de las letras y messagebox para "pop ups") y PIL (Fig. 31).

```
from tkinter import *
from tkinter import ttk, font, messagebox
from PIL import ImageTk, Image
```

Fig. 31: Importar librerías (1)

3.1.2.1. Pantalla principal

Al ejecutar el programa, se abrirá la pantalla principal, cuyo diseño se puede ver más abajo. Se procede a explicar cómo se ha implementado todo el apartado gráfico de la ventana de inicio.

```
root = Tk()
root.title('Acceso a RASPNET')

mainframe = ttk.Frame(root, padding=15, borderwidth=2, relief='raised')
mainframe.grid(column=0, row=0, sticky=(N, W, E, S))

lblTitulo = ttk.Label(mainframe, text='ACCESO A RASPNET', font='Helvetica 30 bold')

imgLib = Image.open('libros.png')
imgLib.thumbnail((250, 250), Image.ANTIALIAS)
imgLibros = ImageTk.PhotoImage(imgLib)
lblImgLibros = ttk.Label(mainframe, image=imgLibros)

lblDatos = ttk.Label(mainframe, text='Introduzca sus datos', font='Helvetica 16')
lblDni = ttk.Label(mainframe, text='DNI', font='Helvetica 16')
lblPin = ttk.Label(mainframe, text='PIN', font='Helvetica 16')

dni = StringVar()
entDni = ttk.Entry(mainframe, textvariable=dni, show='*', font='Helvetica 10')
pin = StringVar()
entPin = ttk.Entry(mainframe, textvariable=pin, show='*', font='Helvetica 10')

btAcceso = ttk.Button(mainframe, text='Acceder', command=guardarDatos)

lblTitulo.grid(column=0, row=0, columnspan=3, sticky=N, padx=5)
lblImgLibros.grid(column=0, row=1, rowspan=6, pady=('20 5'))
lblDatos.grid(column=1, row=1, columnspan=3, sticky=(W, N), padx=20, pady=('10 10'))
lblDni.grid(column=1, row=2, sticky=(W, N), padx=20, pady=0)
lblPin.grid(column=1, row=4, sticky=(W, N), padx=20, pady=0)
entDni.grid(column=1, row=3, sticky=(W, N), padx=20, pady=0)
entPin.grid(column=1, row=5, sticky=(W, N), padx=20, pady=0)
btAcceso.grid(column=2, row=6)
```

Fig. 32: Implementación de la pantalla principal. Interfaz gráfica (1)

La primera línea (Fig. 32) crea la ventana principal, para después darle un nombre. A continuación, se crea el primer *widget*: un *frame*. Se trata básicamente de un marco, un simple rectángulo vacío generalmente usado como contorno de la ventana principal para darle algo de forma o relieve a los bordes. Este se crea mediante el método "Frame()", incluido en la librería ttk, y se indica que estará contenido en la ventana principal "root". Un *frame* puede aceptar varios atributos: "Padding" sirve para que, cuando se implementen otros widgets dentro del marco, estos estén a una distancia de X pixeles del borde, en este caso 15. "Borderwidth" indica que el borde del marco tendrá X pixeles de grosor, en este caso 2. "Relief" sirve para indicar la forma que tendrá el marco, es un aspecto puramente estético; otros relieves posibles son "flat", "solid" o "sunken". Con el método grid se coloca el marco dentro de la ventana. Se verá esto un poco más adelante con más detalle, pero se puede indicar que está en la primera fila y en la primera columna, es decir, al principio de la ventana – las columnas y las filas empiezan a contarse desde 0 –. Con "sticky", se fija el marco en la parte superior, inferior, izquierda y derecha.

A continuación, se empiezan a dibujar todos los *widgets* de la ventana. Estos, como se puede ver, estarán incrustados dentro de "mainframe", el marco que a su vez se había incrustado en la ventana principal. En primer lugar, tenemos un *label*, esto es un *widget* que muestra texto o imágenes. Este primer *label* mostrará el texto de título "ACCESO A RASPNET", que introducirá al usuario a la aplicación. Con el atributo "font", podemos indicar el tipo de fuente, el tamaño y ponerlo en negrita (como es el caso, "bold"), cursiva o subrayado. La pantalla presenta otros tres *displays* de texto: uno que pedirá al usuario que introduzca sus datos, otro que indicará donde escribir el DNI y otro que indicará donde escribir la clave. Los tres están en fuente helvética 16.

El siguiente *label* tiene un poco más de trabajo. En primer lugar, empleando el método "open()" de la biblioteca Image de PIL, se abre una imagen que estará en la misma carpeta que el

programa, en este caso en formato PNG, aunque podría estar en muchos otros formatos. En la siguiente línea, con el comando “thumbnail()”, se modifica el tamaño, en este caso dándole 250x250 píxeles. La opción “ANTIALIAS” es la mejor opción para cuidar la calidad de filtrado al cambiar el tamaño de la imagen. Mediante el método “PhotoImage()” se puede transformar a un formato aceptado por Tkinter, para insertarla dentro de un *widget*. Finalmente, se inserta la imagen dentro del *label* gracias al atributo “image”.

El siguiente *widget* introducido es una *entry*. Se trata de una entrada de texto, un hueco en blanco donde se puede escribir un texto, en este caso, el DNI y la clave de acceso a la Intranet. Lo que se escriba en esta entrada tendrá que ser extraído de alguna manera: para eso tenemos el atributo “textvariable”. Este atributo le dará a las variables “dni” y “pin”, respectivamente, el valor que introduzcamos en la entrada de texto. Para eso, ambas variables han debido ser introducidas previamente como cadenas de texto, gracias a StringVar(). Otro aspecto a destacar es el atributo “show”, gracias al que se permite que no se muestre el texto que tecleamos, y que en su lugar aparezcan asteriscos.

Para acabar con los *widgets*, hay un botón que nos permite confirmar los datos introducidos y acceder a la siguiente pantalla. Con el atributo “command”, indicamos cuál es la función que se ejecutará cuando pulsemos sobre el botón. Este botón llama a la función guardarDatos, que es bastante compleja y se explicará detenidamente en el apartado sobre la funcionalidad.

El siguiente bloque de código, donde se ve en repetidas ocasiones el método grid, sirve para colocar cada *widget* dentro del marco. Antes de empezar a explicar eso, vamos a insertar una captura de la ventana principal que nos permitirá entenderlo mejor (Fig. 33):

Fig. 33: Pantalla principal

Antes de empezar a programar, se ha hecho un esquema a mano de cómo se planea que estén colocados los *widgets*, para ayudar a la hora de escribir el código. Como se puede ver en la figura anterior, tenemos 7 filas (“rows”) y 3 columnas (“columns”). Con los respectivos comandos, se pueden colocar en la rejilla, teniendo en cuenta que se empieza a contar desde la fila y columna

0. Con los atributos “rowspan” y “columnspan” se permite que el widget en cuestión ocupe más de una fila o columna, respectivamente. Tal es el caso de la imagen de los libros que, como bien se indica, empieza en la fila 1 pero se extiende hasta la número 6. Mediante el atributo “sticky”, se puede pegar el *widget* a la izquierda, derecha, parte superior o parte inferior de la columna o fila en la que esté situado, usando W, E, N o S respectivamente. Gracias a “padx” y “pady” se añade la cantidad de píxeles indicada de espacio libre en horizontal o en vertical, respectivamente. Si se indican dos valores distintos en estos parámetros se refieren, respectivamente, a izquierda y derecha en el caso de “padx” o a arriba y abajo en el caso de “pady”.

```
#Las siguientes líneas de código permiten que la ventana se expanda
'''
root.columnconfigure(0, weight=1)
root.rowconfigure(0, weight=1)
mainframe.columnconfigure(0, weight=1)
mainframe.columnconfigure(1, weight=1)
mainframe.columnconfigure(2, weight=1)
mainframe.rowconfigure(0, weight=1)
mainframe.rowconfigure(1, weight=1)
mainframe.rowconfigure(2, weight=1)
mainframe.rowconfigure(3, weight=1)
mainframe.rowconfigure(4, weight=1)
mainframe.rowconfigure(5, weight=1)
mainframe.rowconfigure(6, weight=1)
'''
```

Fig. 34: Implementación de la pantalla principal. Interfaz gráfica (2).

Como pequeño inciso, se explica la utilidad que pueden tener las líneas de código expuestas aquí arriba en la Fig. 34. Gracias al método “columnconfigure” y “rowconfigure” se permite que, al expandir la ventana, la columna y fila respectiva también sigan esta expansión. El atributo “weight” indica, como su propio nombre indica, el peso que tiene cada fila y columna en la expansión; mantener el valor en 1 sirve para que las proporciones se mantengan de la ventana pequeña a la ventana agrandada. En el proyecto se han desactivado estas líneas de código poniéndolas entre tres apóstrofes porque no se quiere que se expanda la ventana al agrandarla, puesto que se ha diseñado para un tamaño concreto y quedaría totalmente desajustada.

```
entDni.focus()
root.bind('<Return>', guardarDatos)

root.mainloop()
```

Fig. 35: Implementación de la pantalla principal. Interfaz gráfica (3).

Para ir acabando, se verán los últimos métodos de la pantalla principal (Fig. 35). En primer lugar, está el método “focus()”, que se aplica a la *entry* “entDni”, la entrada donde se debe introducir el DNI. Este método sirve para que, al ejecutar el programa y abrirse la pantalla principal, el cursor directamente se ponga en la entrada del DNI, para poder empezar a teclear inmediatamente. En cuanto al método “bind()”, sirve para que determinadas pulsaciones de teclas o de ratón puedan ejecutar alguna acción. En este caso concreto ejecutaremos la función guardarDatos, la misma que se ejecuta al hacer click sobre el botón, al pulsar sobre la tecla de “Enter”. Por último, root.mainloop() crea la ventana principal, una vez inicializados y situados todos los *widgets*.

3.1.2.2. Pantalla secundaria

Desde la pantalla principal, se accede a la pantalla secundaria. Esta pantalla secundaria se creará cuando se llame a la función crearVentanaSecundaria. Dentro de está función se encuentran otras funciones a las que se llama más adelante para acceder a las pantallas siguientes.

```
wdw2 = Toplevel()
wdw2.title('RASPNET')

wdw2.grab_set()

secframe = ttk.Frame(wdw2, padding=15, borderwidth=2, relief='raised')
secframe.grid(column=0, row=0, sticky=(N, W, E, S))

lblTitulo2 = ttk.Label(secframe, text='RASPNET', font='Helvetica 30 bold')
lblNombre = ttk.Label(secframe, text=nombreUsuario, font='Helvetica 12')

imgNot = Image.open('books.png')
imgNot.thumbnail((50, 50), Image.ANTIALIAS)
imgNotas = ImageTk.PhotoImage(imgNot)
btNotas = ttk.Button(secframe, text='Notas', image=imgNotas, compound='top',
                     command=lambda : crearVentanaNotas(lAsignaturas, listaNotas))

imgEx = Image.open('pc.png')
imgEx.thumbnail((50, 50), Image.ANTIALIAS)
imgExamenes = ImageTk.PhotoImage(imgEx)
btExamenes = ttk.Button(secframe, text='Exámenes\npoliformat', image=imgExamenes, compound='top',
                     command=crearVentanaExamenes)

btSalir = ttk.Button(secframe, text='Cerrar Sesión', command=cerrarSesion)

lblTitulo2.grid(column=0, row=0, columnspan=2, sticky=N, padx=5)
lblNombre.grid(column=0, row=1, columnspan=2, sticky=E, pady=10)
btExamenes.grid(column=0, row=2, padx=25, pady=25)
btNotas.grid(column=1, row=3, padx=25, pady=25)
btSalir.grid(column=1, row=4, padx=10, pady=10)

wdw2.mainloop()
```

Fig. 36: Implementación de la pantalla secundaria. Interfaz gráfica

Como se puede ver en la Fig. 36, se empieza creando la ventana mediante el comando “Toplevel()”, que sustituye a “Tk()” ya que se trata de una ventana secundaria. En siguiente lugar, con el método “grab_set()” se hace que, desde el momento que se abre esta ventana en primer plano, no se pueda acceder a las otras ventanas abiertas, en este caso la pantalla principal.

El siguiente paso es crear el *frame*, igual que en la ventana principal. Después de esto, hay dos *label*. El primero de ellos muestra el título de la pantalla; en cuanto al segundo, debe mostrar el nombre de usuario, que debe estar guardado en una variable. En el apartado 3.2.2.2 se explica como se obtiene el nombre de usuario al iniciar sesión.

A continuación, se usan los métodos de la librería PIL para subir las imágenes y cambiarles el tamaño, que ya hemos explicado en el apartado anterior. Sin embargo, en este caso las imágenes no van a formar parte de un *label*, sino de un botón. En los parámetros del botón se indica cuál sera la imagen que formará parte de él. Como además de la imagen también hay un texto, necesitamos usar el parámetro “compound” para indicar cómo estará situada la imagen respecto al texto en dicho botón (en este caso, la imagen estará encima del texto, ya que el valor del parámetro “compound” es top”). Como siempre, cada botón lleva un comando asociado, con el objetivo de llamar a una función, para crear las siguientes pantallas. En el caso de “btNotas” se necesita usar la palabra reservada “lambda” porque hay que pasarle dos parámetros a la función crearVentanaNotas a la hora de llamarla. Además de esto, está el botón

"btSalir", usado para cerrar sesión y cuya funcionalidad se explicará también en el apartado 3.2.2.2.

Fig. 37: Pantalla secundaria

En la Fig. 37 se puede ver el diseño de la pantalla (los "grid"). Hay un total de dos columnas y cinco filas.

3.1.2.3. Pantalla de notas

Al hacer click sobre el botón de "Notas" en la pantalla anterior, se llega a la pantalla de notas, donde se pueden ver las notas de toda la carrera. Procedemos a explicar cómo se ha diseñado esta pantalla. Indicar que la creación de esta pantalla corresponde a una función llamada crearVentanaNotas, situada dentro de la función crearVentanaSecundaria.

```
wdwNotas = Toplevel()
wdwNotas.title('NOTAS')

wdwNotas.grab_set()

notasframe = ttk.Frame(wdwNotas, padding=15, borderwidth=2, relief='raised')
notasframe.grid(column=0, row=0, sticky=(N, W, E, S))

lblTituloNotas = ttk.Label(notasframe, text='Notas de la carrera', font='Helvetica 16 bold')
lblElegir = ttk.Label(notasframe, text='Elija una asignatura', font='Helvetica 12')

listaAsignaturas = StringVar(value=lAsignaturas)
lbAsignaturas = Listbox(notasframe, height=12, width=50, listvariable=listaAsignaturas)

for i in range(0,len(lAsignaturas),2):
        lbAsignaturas.itemconfigure(i, background='#f0f0ff')

sbAsignaturas = ttk.Scrollbar(notasframe, orient=VERTICAL, command=lbAsignaturas.yview)
lbAsignaturas.configure(yscrollcommand=sbAsignaturas.set)
lbAsignaturas.selection_set(0)

lblNotaObt = ttk.Label(notasframe, text='Nota:', font='Helvetica 12')

nota = StringVar()
lblNota = ttk.Label(notasframe, textvariable=nota, font='Helvetica 12')

btMostrar = ttk.Button(notasframe, text='Mostrar nota', command=mostrarNota)

lblTituloNotas.grid(column=0, row=0, sticky=(N), pady=10)
lblElegir.grid(column=0, row=1, pady=10)
lbAsignaturas.grid(column=0, row=2, sticky=(N, S, E, W))
sbAsignaturas.grid(column=0, row=2, sticky=(N, S, E))
lblNotaObt.grid(column=0, row=3, sticky=S, pady=10)
lblNota.grid(column=0, row=3, sticky=E, padx=55, pady=10)
btMostrar.grid(column=0, row=4, padx=10, pady=10)

wdwNotas.bind('<Return>', mostrarNota)
lbAsignaturas.bind('<Double-1>', mostrarNota)
```

Fig. 38: Implementación de la pantalla de notas. Interfaz gráfica

Se ve en la Fig. 38 que, una vez más, se crea la ventana con el comando “Toplevel()”, y se le da un título a esta ventana. Luego, está el método “grab_set()”, ya explicado previamente. A continuación se crea el *frame* principal, de la misma forma que se ha hecho anteriormente. Después de esto, vemos los dos primeros *label*: El primero con el título “Notas de la carrera” y el segundo que pide al usuario que elija una asignatura.

Ahora se pasa a trabajar con una lista de parámetros. Se obtiene “lAsignaturas” desde la función guardarDatos, que había sido ejecutada desde la pantalla principal. Se trata de una lista y para insertarla en una *Listbox*, hay que transformarla en string; para eso, se crea una nueva variable string llamada “listaAsignaturas” a partir de “lAsignaturas”. Justo a continuación se crea la *listbox* en cuestión. Esta tiene una altura de 12 palabras, una anchura de 50 caracteres, y toma como variable de lista “listaAsignaturas”, que contendrá todas las asignaturas de la carrera. Después de esto se ejecuta un bucle for que va a iterar entre 0 y la longitud de la lista (es decir, entre todos los valores de la lista de asignaturas) de dos en dos. Sirve simplemente para colorear el fondo, con el atributo “background”, para darle un aspecto más estético.

A continuación, se creará una *scrollbar*, una barra desplazadora para poder navegar entre las distintas asignaturas cuando la cantidad de estas sea mayor de 12 (recordemos que era la altura elegida). Se indica la orientación vertical de la barra con el atributo “orient”, mientras que con el atributo “command” se establece que esta barra se va a aplicar a la *Listbox* de asignaturas que hemos llamado “lbAsignaturas”, en el sentido del eje Y. También se debe configurar la propia *listbox* para que acepte esta *scrollbar*. Para acabar con esto, se establece el primer valor de la lista como predeterminado, para evitar errores.

Después de esto hay dos *labels*: Uno que muestra un texto invariable "Nota:", y otro que mostrará un texto variable. Eso se debe implementar creando la variable string "nota" e asignarla al atributo "textvariable". Para terminar, hay un botón que indica "mostrar nota" y que ejecutará el comando mostrarNota. Así, al pulsar sobre el botón se mostrará la nota de la asignatura que esté seleccionada. En el apartado de la funcionalidad se estudiará cómo está implementada la función mostrarNota.

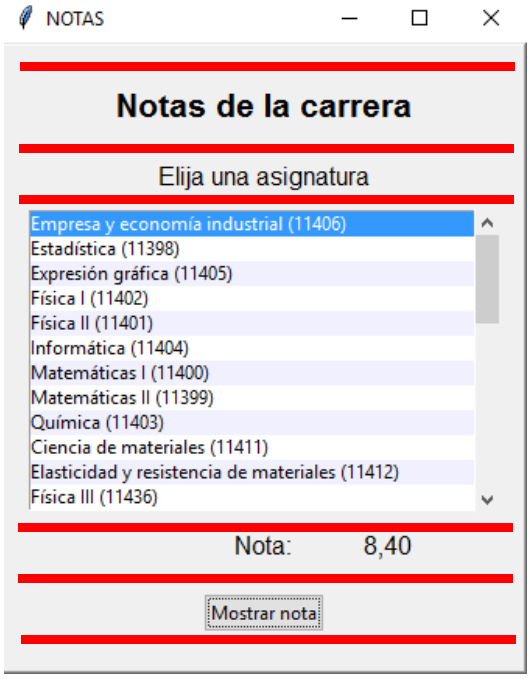


Fig. 39: Pantalla notas

En la Fig. 39 se puede ver el diseño de esta pantalla, donde solo hay una columna y 5 filas, en un diseño bastante simple. Observar que la *Listbox* y la *Scrollbar* deben estar en la misma fila y columna, con los "sticky" adecuados.

Para acabar con el apartado gráfico de esta pantalla, hay dos binding que harán que se ejecute el comando mostrarNota. Uno al pulsar el botón de "Enter" ("<Return>") y otro al hacer doble click ("<Double-1>").

3.1.2.4. Pantalla de exámenes y pantalla de test

Si se hace click sobre el botón "Exámenes Poliformat" en la pantalla secundaria, se llega a la pantalla de exámenes y, posteriormente, a la pantalla de test. Ahora se explicará – brevemente, puesto que se parece mucho a lo que ya hemos explicado en las páginas previas – el código que incumbe a todo el diseño del apartado gráfico. Indicar que la creación de esta pantalla está dentro de una función llamada crearVentanaExámenes, situada a su vez dentro de la función crearVentanaSecundaria.

```
wdwExam = Toplevel()
wdwExam.title('EXÁMENES')

wdwExam.grab_set()

examframe = ttk.Frame(wdwExam, padding=15, borderwidth=2, relief='raised')
examframe.grid(column=0, row=0, sticky=(N, W, E, S))

lblTituloExam = ttk.Label(examframe, text='EXÁMENES', font='Helvetica 30 bold')
lblAsignatura = ttk.Label(examframe, text=asignaturaTest, font='Helvetica 16 bold')
lblTest = ttk.Label(examframe, text='Elija su test', font='Helvetica 12')

listaTest = StringVar(value=lTest)

lbTest = Listbox(examframe, width=30, listvariable=listaTest)
sbTest = ttk.Scrollbar(examframe, orient=VERTICAL, command=lbTest.yview)
lbTest.configure(yscrollcommand=sbTest.set)
lbTest.selection_set(0)

for i in range(0,len(lTest),2):
    lbTest.itemconfigure(i, background='#f0f0ff')

btAcceder = ttk.Button(examframe, text='Empezar test', command=abrirTest)

lblTituloExam.grid(column=0, row=0, sticky=N, padx=5)
lblAsignatura.grid(column=0, row=1, pady=10)
lblTest.grid(column=0, row=2, pady=10)
lbTest.grid(column=0, row=3, sticky=(N, S, E, W))
sbTest.grid(column=0, row=3, sticky=(N, S, E))
btAcceder.grid(column=0, row=4, padx=25, pady=25)

wdwExam.bind('<Return>', abrirTest)
lbTest.bind('<Double-1>', abrirTest)
```

Fig. 40: Implementación de la pantalla de exámenes. Interfaz gráfica

Como se ve en la Fig. 40, este código se parece mucho al visto en el apartado anterior. Antes que nada, se crea la ventana, se le da un título y se bloquean el resto de las ventanas gracias al método “grab_set”. A continuación se crea el marco principal, con los mismos parámetros que en la pantalla anterior. Después, están los tres *label*: el primero que indica el título de la pantalla “EXÁMENES”, el otro que obtiene el nombre de la asignatura, incluido en un fichero, y lo mostrará en la ventana, y un último que pide al usuario que elija un test.

Justo abajo, se empieza a trabajar de nuevo con una lista de parámetros. Se obtiene de un fichero una lista “lTest”, que contiene todos los test disponibles, y que se transformará en una variable string para introducirla en la *listbox*. Dicha *listbox* tendrá una altura estándar de 10 líneas, ya que no se especifica nada, y una anchura de 30 caracteres. También se crea una *scrollbar* vertical, idéntica a la pantalla de notas, se establece el primer valor de la lista como predeterminado y se colorea el fondo alternando entre las líneas de la lista con el bucle for: todo esto es exactamente igual a lo visto anteriormente. Para acabar con el repaso de los *widgets*, se crea un botón que, al ser pulsado, ejecutará la función abrirTest.

En la Fig. 41 se puede ver el diseño de la pantalla, que tiene 5 filas y una columna.

Para acabar con esta pantalla, se han puesto dos binding iguales que en la anterior, para que se ejecute la función abrirTest al pulsar sobre la tecla “Enter” y al hacer doble click sobre un elemento de la lista.

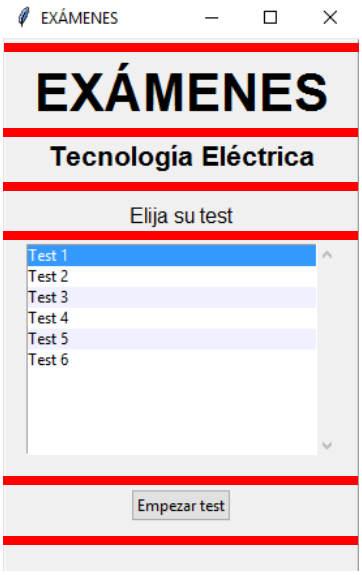


Fig. 41: Pantalla exámenes

Al seleccionar un test, se ejecutará la función abrirTest. Esta función crará una nueva ventana en la que podremos ver las preguntas del test. Estudiemos la creación de esta ventana:

```
wdwTest = Toplevel()
wdwTest.title('TEST')
wdwTest.geometry('600x400')

wdwTest.grab_set()

lblNombreTest = ttk.Label(wdwTest, text=nombreTest, font='Helvetica 16 bold')
lblPregunta = ttk.Label(wdwTest, text=pregunta, font='Helvetica 12')
lblRespA = ttk.Label(wdwTest, text=respuestaA, font='Helvetica 12')
lblRespB = ttk.Label(wdwTest, text=respuestaB, font='Helvetica 12')
lblRespC = ttk.Label(wdwTest, text=respuestaC, font='Helvetica 12')
lblRespD = ttk.Label(wdwTest, text=respuestaD, font='Helvetica 12')

respuesta = StringVar()
rbA = ttk.Radiobutton(wdwTest, text='A.', variable=respuesta, value='A')
rbB = ttk.Radiobutton(wdwTest, text='B.', variable=respuesta, value='B')
rbC = ttk.Radiobutton(wdwTest, text='C.', variable=respuesta, value='C')
rbD = ttk.Radiobutton(wdwTest, text='D.', variable=respuesta, value='D')

imgSiguiente = Image.open('next.png')
imgSiguiente.thumbnail((30, 20), Image.ANTIALIAS)
imgNext = ImageTk.PhotoImage(imgSiguiente)
btNext = ttk.Button(wdwTest, image=imgNext, text='Siguiente', compound='top',
                    command=lambda : siguientePregunta(punt))
btNext.image = imgNext

lblNombreTest.grid(column=0, row=0, columnspan=2, sticky=(N, W), padx=15, pady=10)
lblPregunta.grid(column=1, row=1, sticky=W, padx=15, pady=15)
rbA.grid(column=0, row=2, sticky=(W, N), padx=5, pady=5)
rbB.grid(column=0, row=3, sticky=(W, N), padx=5, pady=5)
rbC.grid(column=0, row=4, sticky=(W, N), padx=5, pady=5)
rbD.grid(column=0, row=5, sticky=(W, N), padx=5, pady=5)
lblRespA.grid(column=1, row=2, sticky=(W, N), padx=5, pady=5)
lblRespB.grid(column=1, row=3, sticky=(W, N), padx=5, pady=5)
lblRespC.grid(column=1, row=4, sticky=(W, N), padx=5, pady=5)
lblRespD.grid(column=1, row=5, sticky=(W, N), padx=5, pady=5)
btNext.grid(column=2, row=6, sticky=(N, E), padx=5, pady=5)
```

Fig. 42: Implementación de la pantalla de test. Interfaz gráfica

En primer lugar, como siempre, se crea la ventana (Fig. 42). Habitualmente, el tamaño de la ventana dependía de los *widgets* presentes en ella y de su distribución, pero en este caso, se establece un tamaño de ventana preciso, de 600x400. También se vuelve a incluir el método "grab_set", del que ya se ha hablado más de una vez.

Hay un total de 6 *label*. El primero, de mayor tamaño y en negrita, mostrará el nombre del test, el siguiente mostrará la primera pregunta y los cuatro siguientes mostrarán las 4 posibles respuestas. El contenido del texto se extrae de un fichero, más adelante veremos cómo se hace.

Lo siguiente son los 4 *radiobutton* desde los cuales se debe marcar la respuesta que se considere correcta. Estos son pequeños círculos en los que sólo puede estar marcado uno de todos los que hay. Dependiendo de cuál se marque, la variable string "respuesta" tomará un valor u otro.

Después hay un botón con imagen y texto "siguiente", que al ser pulsado ejecutará el comando siguientePregunta. Ya se ha explicado cómo insertar una imagen en un botón y para qué se usa la palabra reservada "lambda", por lo que no se perderá más tiempo con esto.

A continuación, se colocan los *widgets* en filas y columnas. En la Fig. 44 se puede ver cómo se ha hecho esto.

Dentro de la función abrirTest no sólo está la creación de la pantalla de test, sino que también encontramos un cronómetro, que debe mostrar cuanto tiempo nos queda (Fig. 43). Como peculiaridad, aquí se usa el método "pack" en vez de "grid". Esto es porque este contiene los parámetros "fill" y "expand" que permitirán que el fondo de pantalla de color negro ocupe toda la pantalla del cronómetro. En la Fig. 44 se muestra el aspecto que presenta este temporizador.

```
wdwCrono = Toplevel()
wdwCrono.title('CRONÓMETRO')

lblClock = ttk.Label(wdwCrono, font='ubuntu 30 bold', background='#3C3B37', foreground='white')
lblClock.pack(fill=BOTH, expand=1)
```

Fig. 43: Implementación del cronómetro. Interfaz gráfica

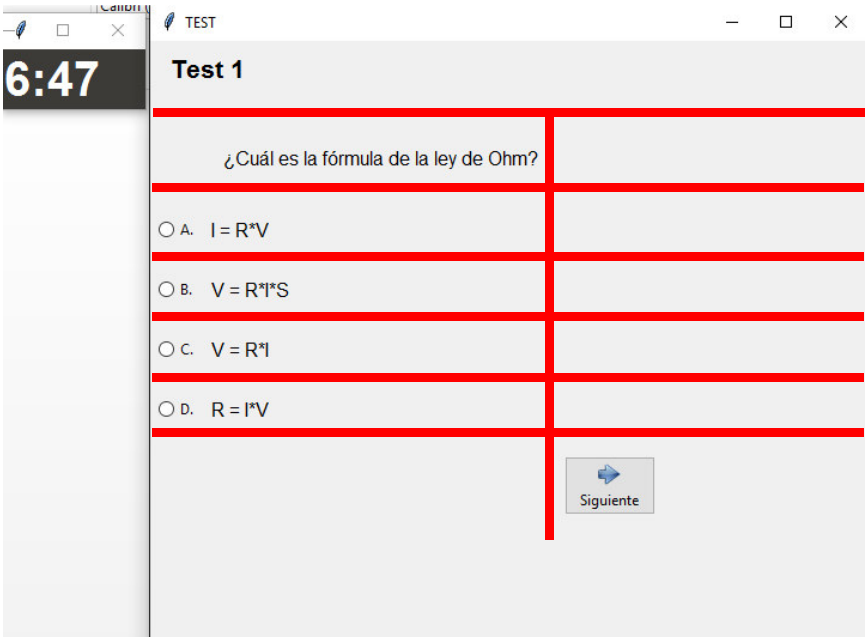


Fig. 44: Pantalla test. Cuenta atrás.

3.2. Funcionalidad

3.2.1. Librerías utilizadas

3.2.1.1. Requests

Requests es una librería HTTP, creada para sustituir a otras librerías como urllib, que podía realizar funciones parecidas pero que, según los creadores de requests, había quedado obsoleta y era poco "pythonica". Requests permite hacer todas estas funciones de forma más sencilla. Esta librería requiere instalación previa, puesto que no se encuentra incluida en Python. Se puede encontrar más información en su página web oficial: http://docs.python-requests.org/en/master/.

Básicamente, esta librería sirve para interactuar con Internet. Permite obtener información de URL de la web, enviar información a la web, crear sesiones con Cookies persistentes, establecer tiempos de espera de conexión, etc. Concretamente, aquí ha sido utilizada para obtener información de la Intranet y a la hora de hacer el login en UPVNET y mantener esta sesión.

3.2.1.2. Http.cookiejar

Http.cookiejar es una librería, ya incluida en Python, que permite generar y mantener cookies persistentes automáticamente. Las cookies son pequeños datos enviados por un servidor web y almacenados por el usuario, de manera que el sitio web pueda consultar la actividad realizada previamente por el usuario. Una de las funciones principales de las cookies es almacenar los datos de usuario y contraseña, cuando ingresamos en una página web, y así no necesitar ingresarlos a cada página nueva del servidor que visitemos. El tema de las cookies también genera cierta controversia, dado que estas pueden obtener información sobre los hábitos de navegación de los usuarios, lo que puede provocar problemas de privacidad, sobre todo cuando individuos con intenciones poco amigables pretenden hacer uso de esta información. Sin embargo, esto no es relevante para este proyecto, aunque está bien mencionarlo para tener una visión más amplia de lo que son las cookies.

Aquí se usa esta librería para lo que hemos comentado más arriba. Combinado con los requests se puede hacer que una vez iniciada la sesión en UPVNET, mediante el uso de cookies, esta se mantenga, permitiendo navegar por la página sin necesidad de ingresar los datos nuevamente.

3.2.1.3. BeautifulSoup

BeautifulSoup es una biblioteca para parsear documentos HTML. Parsear significa recorrer una base de datos para extraer de ahí la información deseada. En cuanto a HTML, siglas de *HyperText Markup Language*, es uno de los lenguajes más usados para crear páginas web. No se entrará a comentar más cosas sobre este lenguaje, puesto que no es el objetivo de este TFG, pero es importante saber que casi todo lo que podemos ver en una página web está programado en este lenguaje.

Así pues, BeautifulSoup es muy utilizado para lo que se conoce como *web scraping*. El *web scraping* es una técnica utilizada para extraer datos de una página web. De cierta forma, se relaciona con la indexación de la web, utilizada por los motores de búsqueda, pero realmente

está más enfocado en recoger datos sin estructura de la web, como en formato HTML, y transformarlos en datos estructurados.

La versión más reciente de BeautifulSoup es la 4. Es una librería totalmente "pythonica", relativamente fácil de aprender y uno de los parsers más populares de Python. Su nombre viene de una canción del popular cuento *Alicia en el país de la Maravillas.*

En el proyecto para recolectar datos de interés de UPVNET, como puede ser el nombre de usuario. Una vez más, se usará combinado con los requests: estos permiten obtener la URL deseada, y mediante BeautifulSoup se obtiene el código HTML y podremos obtener de él lo que queramos.

3.2.2. Código

Ya se ha explicado previamente la creación de toda la interfaz gráfica de la aplicación, por lo que ahora procedese procederá a contar cómo se ha programado toda la funcionalidad de la aplicación, para conseguir que haga lo deseado desde el principio. Conviene recordar que, aunque se pegarán los trozos de código necesarios para explicarlo todo, se puede consultar al completo en el anexo.

Igual que para implementar la parte gráfica de la aplicación se han debido importar las librerías gráficas, ahora hay que importar las librerías necesarias para la funcionalidad del programa: requests, http.cookiejar y BeautifulSoup (Fig. 45).

```
import requests
import http.cookiejar
from bs4 import BeautifulSoup
```

Fig. 45: Importar librerías (2)

3.2.2.1. Pantalla principal

La función clave en el desarrollo de nuestro proyecto y en el acceso a UPVNET es la función guardarDatos que, como ya se ha visto antes, se ejecuta al pulsar sobre el botón "acceder" en la pantalla principal. Esta función se ocupa de, entre otras cosas, obtener los datos introducidos en esta pantalla para pasarlas a la página de la UPV, para acceder a ella. Se comentarán a continuación las líneas de código que implican el acceso a la intranet (Fig. 46 y 48) – quedarán otras por explicar que se usan en otras pantallas:

```
def guardarDatos(*args):
    with requests.Session() as s:

        jar = http.cookiejar.CookieJar()

        #Links URL
        login_url  = 'https://intranet.upv.es/exp/aute_intranet'
        inicio_url = 'https://intranet.upv.es/pls/soalu/sic_menu.MiUPV?P_IDIOMA=c&P_MODO=alumno'
        nombre_url = 'https://intranet.upv.es/pls/soalu/sic_menu.Alumno?P_IDIOMA=c'
        notas_url  = 'https://intranet.upv.es/pls/soalu/sic_asi.notes_temaalu_asi'

        datos_login = {
                       'dni': dni.get(),
                       'clau': pin.get()
                       }

        datos_notas = {'p_curso' : '1'}

        s.get(login_url, cookies=jar)
        s.post(login_url, data=datos_login, cookies=jar)

        entDni.delete(0, 'end')
        entPin.delete(0, 'end')
```

Fig. 46: Implementación de la pantalla principal. Funcionalidad (1)

La función guardarDatos recibe como argumento "*args". Esto es realmente una llamada a una lista arbitraria de elementos, y en el caso actual es necesario para indicar que se pasa como variable los binding realizados, en este caso, el binding que hace que al pulsar sobre el botón "Intro" se ejecute esta función.

Nada más empezar, gracias a la librería requests se crea una sesión, que se llamará "s". Una sesión sirve para que una vez esté hecho el login en la página de la UPV no sea necesario volver a ingresar los datos cada vez que se quiera hacer una operación en páginas del mismo dominio. Esta función la realiza junto a las cookies. Como se verá, habrá que especificar cada vez que se acceda a una URL que esta va a formar parte de la sesión "s". A partir de la siguiente línea se pone una identación, que se mantendrá todo el resto de la función, puesto que toda ella se implica a la misma sesión.

En la siguiente línea, se llama a la biblioteca http.cookiejar, y dentro de ella a la clase CookieJar. Lo que hace esta clase es almacenar automáticamente cookies de direcciones web cuando sea requerido y devolverlas cuando se solicite. En este caso se almacena en la variable "jar". Ya se ha hablado antes de la importancia de las cookies para que no sea necesario ingresar los datos de acceso constantemente, algo que es clave en este programa y que queda solventado gracias a las cookies y las sesiones.

Después, se almacenan las distintas URL necesarias; algunas se usarán en esta pantalla, mientras que otras se emplearán más adelante. Siguiendo con los primeros pasos de esta función, se obtiene de las *entry* "dni" y "pin" el DNI y la clave ingresados por el alumno, y se guardan en un diccionario que tiene como claves "dni" y "clau". El porqué de estas claves se puede obtener desde la página de la Intranet. Si, al hacer un login, pulsamos la combinación de teclas ctrl + Shift + I, podemos acceder a la consola para inspeccionar la página. Aquí se puede ver todo el código fuente, inspeccionar los distintos elementos y muchas otras cosas. Si en el apartado "Network" se busca "aute_intranet" -> "Headers" -> "Form data", podemos observar los distintos parámetros que se le pueden devolver a la página (Fig. 47):

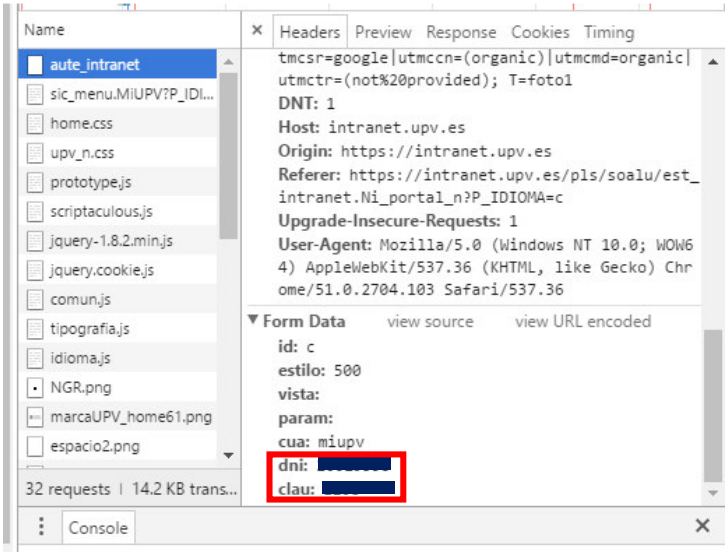


Fig. 47: Inspeccionar la pantalla de ingreso a la Intranet. Datos a enviar

Una vez conocido a eso, en primer lugar se obtienen la URL de autentificación a la intranet y las cookies asociadas a ellas gracias al método "get" de requests. Y en segundo lugar se devuelven los datos de login junto a las cookies, para completar el login a la intranet. Mencionar que, evidentemente, todo esto está incluido en la sesión "s".

Una vez realizado este login, se borran los datos ingresados en las entradas de dni y pin, para en el caso de que se quiera hacer otro login posterior.

```
req = s.get(inicio_url)

if req.headers['Content-Length']=='11858':
    messagebox.showinfo(message='DNI o PIN incorrecto. Por favor, revise sus datos', parent=mainframe)
    entDni.focus()

else:
    buscNombre = s.get(nombre_url)
    htmlNombre = BeautifulSoup(buscNombre.text, 'html.parser')
    nombre = htmlNombre.find('a', {'name' : 'panel_155'}).getText()

    notasUPV = s.get(notas_url, cookies=jar)
    notasUPV = s.post(notas_url, cookies=jar, data=datos_notas)
    htmlNotas = BeautifulSoup(notasUPV.text, 'html.parser')

    listAsignaturas = []

    table = htmlNotas.find('table', {'class' : 'upv_listacolumnas'})

    for asignatura in table.find_all('td', {'class' : 'upv_listacolumnassubtitulo'}):
        listAsignaturas.append(asignatura.getText())

    listDatos = []
    listNotas = []
    rango     = range(len(listAsignaturas))
    iterAsig  = list(rango)

    for datosAsignatura in table.find_all('tr', {'class' : 'upv_listanon'}):
        for nota in datosAsignatura.find_all('td'):
            listDatos.append(nota.getText())

    for i in iterAsig:
        listNotas.append(listDatos[2+6*i])

    messagebox.showinfo(message='Datos correctos. Accediendo a Poliformat', parent=mainframe)
    entDni.config(state='disabled')
    entPin.config(state='disabled')
    crearVentanaSecundaria(nombre, listAsignaturas, listNotas)
```

Fig. 48: Implementación de la pantalla principal. Funcionalidad (2)

El siguiente paso es obtener la información de la URL de inicio de la Intranet y guardarla en una variable llamada "req". Respecto a esto, hay que fijarse de nuevo en la página de "Inspeccionar", en el apartado "Response Headers", donde al ingresar un dni o una clave incorrecta el valor de

la variable “Content-Length” toma el valor 11858 (Fig. 49), algo que no es así si el ingreso se realiza correctamente. Por eso, esta variable va a ser la que sirva para verificar si los datos ingresados son correctos o no.

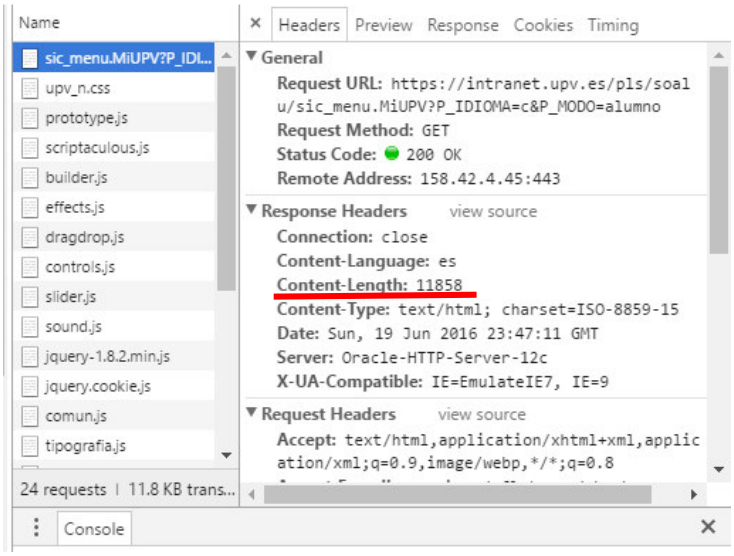


Fig. 49: Response headers de la página inicial de la UPV

En el caso de que los datos ingresados sean incorrectos, se abrirá una ventana con un mensaje que lo indicará y se volverá a focalizar el puntero en la *entry* del DNI, para ingresar de nuevo los datos. En caso contrario, después de hacer una serie de operaciones de las que se hablará más adelante, ya que no tienen más interés en lo que respecta el acceso a la intranet, se abrirá una ventana con un mensaje que dirá que los datos ingresados son correctos. Las entradas de DNI y pin se bloquearán para impedir que se realice un nuevo login mientras la sesión siga abierta. Y, finalmente, se ejecutará la función crearVentanaSecundaria, de cuya implementación gráfica ya se ha hablado previamente. Los tres parámetros que se le envían a esta función serán mencionados en las próximas páginas.

3.2.2.2. Pantalla secundaria

La pantalla secundaria funciona como intermediaria entre la pantalla principal, donde se ingresan los datos, con las otras pantallas, donde se encuentran las auténticas aplicaciones importantes del programa. Por lo tanto, hay poca implementación que comentar, puesto que esta pantalla es básicamente, como ya se ha visto al hablar de su interfaz gráfica, un conjunto de iconos que ejecutan otras funciones incluidas dentro de crearVentanaSecundaria y que abren nuevas ventanas. Aun así, sabiendo que en la pantalla secundaria se muestra el nombre de usuario de la Intranet, se procederá a explicar cómo se obtiene.

Para eso, se debe volver a la Fig. 48. En ella, se veía que cuando al ejecutar el código incluido en la condición “else”, es decir, cuando los datos ingresados eran correctos, se ejecutan una serie de operaciones con el objetivo de extraer el nombre de la Intranet. El primer paso es, mediante el método de requests “get”, obtener la URL de la Intranet donde podemos encontrar el nombre de usuario. A continuación, mediante el método “text” de la librería BeautifulSoup, que sirve para hacer *web scraping* como ya hemos comentado, se extrae todo el código HTML de esta URL

y se guarda en una variable “htmlNotas”. “html.parser” sirve para facilitar el “parsing” y que se pueda acceder fácilmente al código HTML. Por último, dentro de todo el código HTML se tiene que buscar donde está el nombre del alumno. Para eso se puede seleccionar el nombre del alumno en la página web, hacer click derecho y seleccionar “Inspeccionar”. En el apartado “Elements” se puede ver entre qué etiquetas se encuentra el contenido. En este caso, está entre las etiquetas “a” con el nombre “panel_155” (Fig. 50). Una vez buscado, este contenido se extrae con el método “getText()” y se guarda en la variable “nombre”, para luego pasarlo como parámetro cuando se ejecute la función crearVentanaSecundaria.

Fig. 50: Código HTML de la página de la Intranet. Obtener nombre de usuario

Además de esto, en la pantalla secundaria también tenemos el botón de cerrar sesión, cuya pulsación ejecutará la función cerrarSesión, que podemos ver en la Fig. 51:

```
def crearVentanaSecundaria(nombreUsuario, lAsignaturas, listaNotas):

    def cerrarSesion():
        wdw2.destroy()
        entDni.config(state='enabled')
        entPin.config(state='enabled')
        entDni.focus()
```

Fig. 51: Implementación de la pantalla secundaria. Cerrar sesión

Lo que hace esta función es muy simple. Simplemente destruye la ventana secundaria, volviendo a dejar únicamente la ventana principal, y en esta vuelve a activar las entradas de DNI y pin, para poder volver a iniciar una nueva sesión. No es necesario hacer nada más, porque una vez se ha salido de la pantalla secundaria ya no se pueden realizar más operaciones con el usuario ingresado previamente, puesto que se ha acabado la sesión “s”. Cuando se introduzcan los nuevos datos se iniciará una nueva sesión de requests.

3.2.2.3. Pantalla de notas

Al pulsar sobre el botón correspondiente, se ejecuta la función crearVentanaNotas, de cuyo apartado gráfico ya se ha hablado anteriormente. Ahora se procederá a comentar su funcionalidad.

Para eso, lo primero que se debe hacer es volver a la Fig. 48. Una vez ya se tiene el nombre de usuario, hay que obtener mediante "get" la información de una nueva URL – aquella donde están todas las notas – y sus cookies asociadas, para justo después devolver estas cookies y el diccionario "datos_notas", que se puede ver en la Fig. 46, gracias al método "post". Al enviar en la clave "p_curso" el valor "1" (Fig. 52) se está diciendo que se quiere ver las notas de toda la carrera, en vez de ver sólo las del curso actual – para lo que habría que enviar el valor "2". Esta información se obtiene de la misma manera que se obtenían previamente los datos de ingreso necesarios.

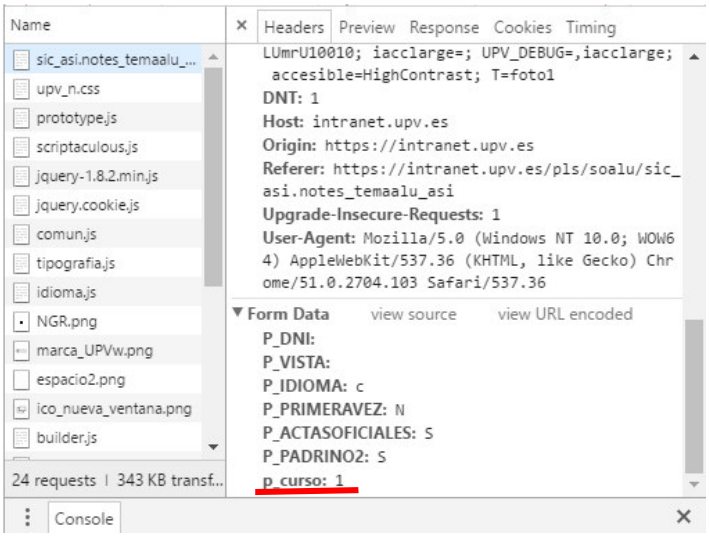


Fig. 52: Inspeccionar la pantalla de notas de la Intranet. Datos a enviar

Una vez enviada esta información, se extrae igual que antes todo el código HTML de esta página y se guarda en la variable "htmlNotas", para proceder a continuación a extraer la información que interesa, aunque en este caso será algo más complejo que cuando se buscaba el nombre de usuario. En "Inspeccionar" -> "Elements" hay que buscar dónde está la tabla de notas que interesa. Se puede observar que se encuentra entre las etiquetas "table" con la clase "upv_listacolumnas" (Fig. 53). Esta tabla se guarda en una variable llamada "tabla".

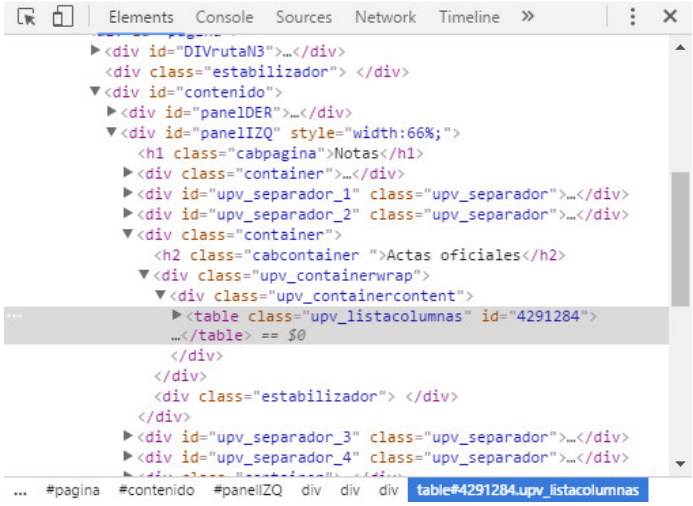


Fig. 53: Código HTML de la página de notas. Obtener tabla de notas

A continuación, se crea una lista vacía llamada “listAsignaturas”, donde se guardarán todas las asignaturas. En la tabla anterior, se buscan todos los valores que se encuentren entre las etiquetas “td” con la clase “upv_listacolumnassubtitulo” (Fig. 54), mediante el método “Find_all”. Mediante un bucle for, se itera entre todos estos valores, se obtiene el nombre de cada asignatura y se guarda en la lista correspondiente. Esta lista se enviará como parámetro al ejecutar la función crearVentanaSecundaria, y también se obtendrá en la “sub-función” crearVentanaNotas. Recordar que esta lista se transformará en una variable string para incluirla en la *listbox* correspondiente.

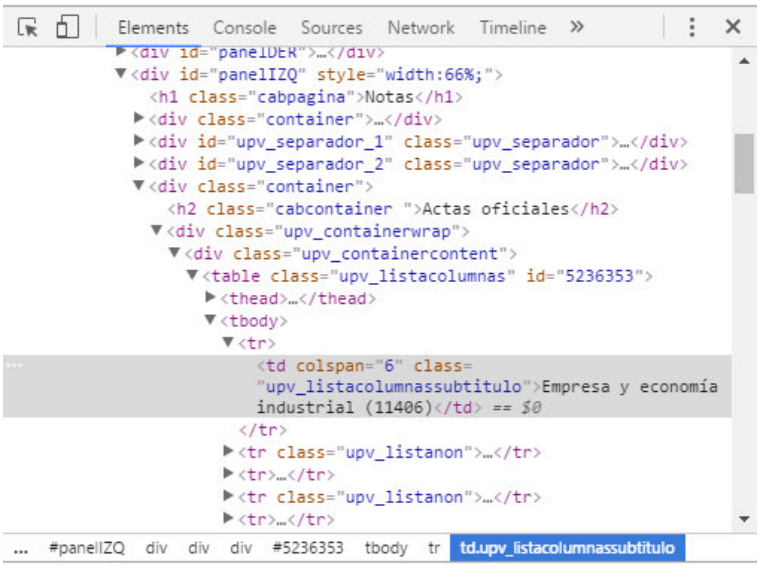


Fig. 54: Código HTML de la página de notas. Obtener asignaturas

Después de obtener las asignaturas, el siguiente paso será obtener las notas. Para eso, se empieza creando dos listas vacías. Después, se obtiene la longitud de la lista de asignaturas para crear un rango a partir de este número. Range es, en Python 3, un tipo de dato, aunque se utiliza como una función. Lo que hace es crear una serie de números enteros que se suceden de uno en uno, empezando desde el 0. Una vez se usa el tipo “range” se debe transformar en una lista que servirá para iterar más adelante, ya que contiene tantos parámetros como asignaturas hay.

Una vez hecho esto, se puede volver a iterar en la tabla extraída más arriba mediante un bucle for: se obtiene mediante el método find_all los datos que estén entre las etiquetas “tr”, con la clase “upv_listanon”. Aquí, como se puede ver en la Fig.55, no está sólo la información de las notas, sino que hay otras cosas como el año, la convocatoria, la calificación, el estado y comentarios especiales. Para lo que se está desarrollando sólo interesan las notas, pero para extraerlas se deben obtener también, en un principio, los otros datos. Por eso, dentro del último bucle for se ejecuta otro bucle for donde se buscan todas las etiquetas “td”, para guardar su contenido en el vector “listDatos”.

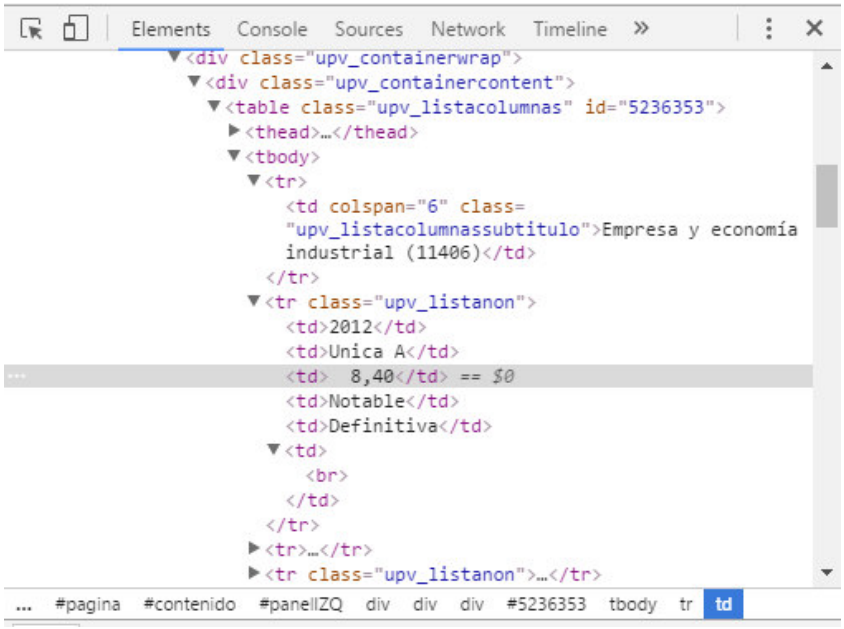


Fig. 55: Código HTML de la página de notas. Obtener notas

En la lista de datos, se tienen las notas a partir del tercer parámetro y en intervalos de 6 en 6. Por eso, en el siguiente bucle se itera en la lista "iterAsig", que tiene un número por cada asignatura que haya, obteniendo los valores de la lista de datos de 6 en 6. Así, se puede guardar en una lista "listNotas" todas las notas, en orden, de las distintas asignaturas que ya se habían extraído en la lista "listAsignaturas". Al igual que su homóloga, "listNotas" se enviará como parámetro al ejecutar la función crearVentanaSecundaria, y también se obtendrá en la "sub-función" crearVentanaNotas.

Además, en la función que crea la ventana de notas hay otra función más que se ejecuta al pulsar sobre el botón "mostrar nota": la función mostrarNota (Fig. 56).

```
def crearVentanaNotas(lAsignaturas, listaNotas):

    def mostrarNota (*args):
        lbSeleccion = lbAsignaturas.curselection()
        numAsig = int(lbSeleccion[0])
        notaAsig = listaNotas[numAsig]
        nota.set(notaAsig)
```

Fig. 56: Implementación de la pantalla de notas. Mostrar nota

Mediante el método "curselection()", asociado a las Listbox, se obtiene el valor seleccionado en esta. Es decir, gracias a este método se puede averiguar cuál es la asignatura seleccionada por el usuario. Sin embargo, esto no devuelve un entero, sino una tupla en cuyo interior está el valor numérico, empezando por 0, de la selección elegida. Por eso, se debe transformar a entero el valor de esta tupla y se guarda en una variable. Conocido el número de la asignatura elegida, se puede obtener la nota asociada a dicha asignatura buscándola en la lista correspondiente. Una vez extraída la nota, simplemente se modifica el valor de la variable de texto "nota", incluida dentro de un label, gracias al método set().

3.2.2.4. Pantalla de exámenes y pantalla de test

Al pulsar sobre el botón correspondiente, se ejecuta la función crearVentanaExámenes, de cuyo apartado gráfico ya se ha hablado anteriormente. Ahora se va a proceder a comentar su funcionalidad.

```
fichero_listaTest = 'Lista Test.txt'
archivo_listaTest = open(fichero_listaTest, 'r')

asignaturaTest = archivo_listaTest.readline().strip()

lTest0 = []
lTest = []

for linea in archivo_listaTest:
    lTest0.append(linea)

for i in lTest0:
    lTest.append(i.strip())

archivo_listaTest.close()
```

Fig. 57: Implementación de la pantalla de exámenes

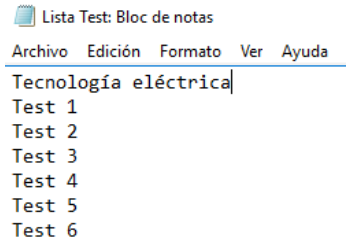


Fig. 58: Fichero "Lista Test.txt"

En la misma carpeta donde esté el programa, es necesario tener un fichero de texto similar al que vemos en la Fig. 58. La primera línea del fichero mostrará el nombre de la asignatura y las líneas posteriores, el nombre de cada Test disponible. Este fichero se abre desde el programa en modo de lectura ("r"). Lo primero que se hace, como se ve en la Fig. 57, es extraer el nombre de la asignatura leyendo la primera línea del fichero, gracias al método "readline()". Este método tiene en cuenta el salto de línea "\n" del fichero, por lo que hay que quitarlo con "strip()". A continuación se crean dos listas vacías. La primera será una lista auxiliar donde se irá iterando línea por línea y guardando el nombre de cada test. En el segundo bucle, se itera esta primera lista auxiliar para guardar en la lista definitiva "lTest" los nombres de los test sin el salto de línea. Una vez hechas todas las operaciones, se cierra el fichero de texto con el método "close()".

```
lbSelect = lbTest.curselection()

#Los nombres deben ser cambiados por el profesor
fichero_Test1 = 'Test 1.txt'
fichero_Test2 = 'Test 2.txt'

if lbSelect[0] == 0:
    archivo_Test = open(fichero_Test1, 'r')
elif lbSelect[0] == 1:
    archivo_Test = open(fichero_Test2, 'r')

ptsxPregunta = float(archivo_Test.readline())
ptsxFallo = float(archivo_Test.readline())
nombreTest = archivo_Test.readline().strip()
tiempoTest = int(archivo_Test.readline())

archivo_Test.readline()

punt = []

pregunta = archivo_Test.readline().strip()
respuestaA = archivo_Test.readline().strip()
respuestaB = archivo_Test.readline().strip()
respuestaC = archivo_Test.readline().strip()
respuestaD = archivo_Test.readline().strip()
```

Fig. 59: Implementación de la pantalla de test (1)

```
countdown(60*tiempoTest)

wdwExam.destroy()
```

Fig. 60: Implementación de la pantalla de test (2)

Como se ha visto, una vez seleccionado un test y pulsado el botón “Empezar test” (o, en su defecto, al hacer doble click o pulsar sobre “Intro”), se ejecutará la función abrirTest (Fig. 59). Ya se vio en qué consiste la creación de la interfaz gráfica de un test, pero ahora se va a estudiar cómo se ha desarrollado su funcionalidad, que también requiere del uso de ficheros. Para cada opción de test disponible en la lista de la pantalla anterior, se debe añadir el fichero correspondiente, para poder acceder a él. También habría que modificar el bucle “if” para que, dependiendo de la selección realizada en la *listbox*, se abra un test u otro, mediante el método de ficheros “open()”. Una vez abierto el fichero correspondiente, podemos empezar a trabajar con él y extraer la información. En la Fig. 61 se puede ver la apariencia que tienen estos ficheros:

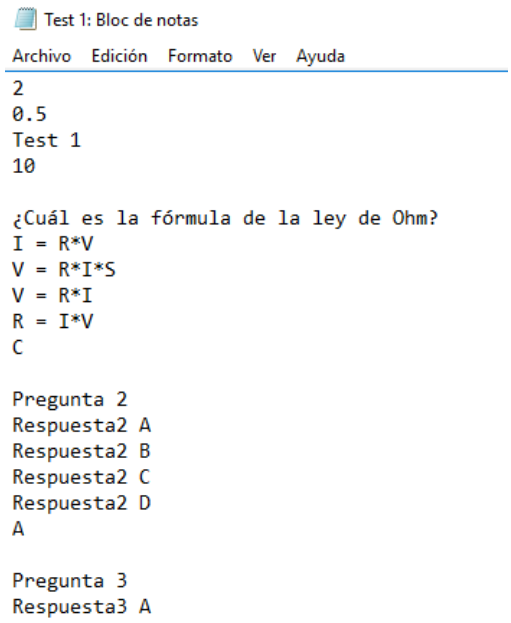


Fig. 61: Fichero “Test 1.txt”

Mediante el método “readline()” se pueden empezar a leer las 4 primeras líneas del fichero. En la primera línea se obtiene la puntuación obtenida por cada respuesta correcta. Como desde el fichero se lee un string, es necesario transformarlo a float. La segunda línea indica cuántos puntos se pierden por cada respuesta incorrecta. La tercera línea muestra el nombre del test, que ya venía incluido en la *listbox* de la ventana anterior pero que resulta más cómodo añadir en el fichero. Por último, el último número indica, en minutos, el tiempo de duración del test. Se vuelve a ejecutar el método “readline()” una vez más para saltar la línea en blanco, sin guardarla en ninguna variable.

Una vez hecho esto hay que crear una lista llamada “punt”, que se pasará como parámetro cada vez que se pulse sobre el botón “Siguiente” y se ejecute la función siguientePregunta, actualizándose cada vez que se haga dicha acción. A continuación, se vuelven a leer las siguientes líneas del fichero: la primera línea se guardará en la variable “pregunta” y las cuatro siguientes en la variable de la respuesta correspondiente, en orden alfabético de A a D. Cada una de estas variables se mostrará en el Label correspondiente.

Por último, se debe realizar dos acciones, que se pueden ver en la Fig. 60: la primera consiste en llamar a la función countdown. La pantalla del cronómetro ya se ha creado al ejecutarse la

función abrirTest, pero la cuenta atrás todavía no está en marcha. Para eso está la función countdown, cuya implementación se puede ver en la Fig. 62 y que se explicará en el próximo párrafo. La segunda acción es simplemente cerrar la ventana de exámenes, puesto que una vez abierto el test seleccionado no se necesita para nada.

```
def countdown(secs):

    if secs >= 0:
        if wdwCrono.state() == 'normal':
            minutos = secs//60
            segundos = secs - 60*minutos
            tiempo = str(minutos) + ":" + str(segundos).zfill(2)

            root.after(1000, countdown, secs-1)
            lblClock['text'] = tiempo

    else:
        messagebox.showinfo(message='Tiempo finalizado. Cerrando test', parent=wdwTest)
        wdwTest.destroy()
        wdwCrono.destroy()
```

Fig. 62: Implementación de la cuenta atrás.

Al llamar a la función countdown esta recibe como valor el tiempo (en minutos) de duración del examen multiplicado por 60. Es decir, el valor “secs” corresponde al tiempo en segundos que debe durar el test. En esta función, lo primero que se hace es comprobar que el valor de “secs” es mayor o igual a 0, es decir, que la cuenta atrás todavía no ha finalizado. Dentro de este condicional hay otro que comprueba que la ventana del cronómetro sigue en marcha, gracias al método state(), que devuelve el estado de la ventana, que debería ser “normal” si sigue abierta. Esta comprobación es necesaria para que si el alumno finaliza el test antes de que acabe el tiempo, cerrando así el cronómetro, deje de ejecutarse la cuenta atrás. Dentro de los dos condicionales, se obtiene el tiempo en minutos, como truncamiento de la división de los segundos totales entre 60, y el tiempo en segundos. Estos valores se pasan a string y se guardan dentro de la variable tiempo, que a su vez se insertará dentro del único Label existente en la ventana del cronómetro. Cabe destacar el método zfill(2), que sirve para que los segundos se muestren siempre con dos cifras, incluidos los números situados entre el 0 y el 9 (mostrar, por ejemplo 12:07 en vez de 12:7). Una vez hecho todo esto, con el método root.after se hace que cada 1000 milisegundos – cada segundo – se vuelva a ejecutar la función countdown, restándole a cada ciclo 1 al valor de secs, para volver a hacer todo lo anterior con el nuevo valor.

En caso de que secs sea menor que 0, significará que el tiempo se ha acabado, por lo que ya no tiene sentido seguir manteniendo la marcha atrás y se ejecutarán las líneas de código incluidas dentro del “else”. Concretamente, se abrirá un mensaje en una ventana que indicará al alumno que el tiempo está finalizado y que se va a proceder a cerrar el test, para después destruir tanto la ventana del test como la ventana del cronómetro.

En la pantalla de test, al pulsar sobre el botón de “siguiente” se ejecuta la función siguientePregunta, cuyo objetivo es, en primer lugar, guardar la respuesta dada por el usuario y, en segundo lugar, actualizar las preguntas y respuestas para mostrar la siguiente cuestión. Este botón estará disponible desde que se abre la ventana hasta que se cierre el test, ya sea porque se ha acabado el tiempo o porque se ha completado el examen. En la Fig. 63 se puede ver la implementación de esta función.

```
respuestaCorrecta = archivo_Test.readline().strip()

if archivo_Test.readline() != '':

    if respuesta.get() == respuestaCorrecta:
        punt.append(ptsxPregunta)
    elif respuesta.get() == '':
        punt.append(0.0)
    else:
        punt.append(-ptsxFallo)

    pregunta = archivo_Test.readline().strip()
    respuestaA = archivo_Test.readline().strip()
    respuestaB = archivo_Test.readline().strip()
    respuestaC = archivo_Test.readline().strip()
    respuestaD = archivo_Test.readline().strip()

    respuesta.set('')

    lblPregunta['text'] = pregunta
    lblRespA['text'] = respuestaA
    lblRespB['text'] = respuestaB
    lblRespC['text'] = respuestaC
    lblRespD['text'] = respuestaD

else:
    if respuesta.get() == respuestaCorrecta:
        punt.append(ptsxPregunta)
    elif respuesta.get() == '':
        punt.append(0.0)
    else:
        punt.append(-ptsxFallo)

    archivo_Test.close()
    messagebox.showinfo(message='Test finalizado. Guardando datos', parent=wdwTest)
    guardarPuntuacion()
    wdwCrono.destroy()
    wdwTest.destroy()
```

Fig. 63: Implementación de la función siguientePregunta

Si se recuerdan las Fig. 59 y 61, se puede ver que se ha leído la primera pregunta y las 4 respuestas. La siguiente línea en el fichero muestra la respuesta correcta (A, B, C o D), y lo primero que se hace es leerla y guardar su valor en una variable “respuestaCorrecta”.

Una vez hecho esto, el siguiente paso es leer la siguiente línea, y comprobar si se trata de una cadena vacía. En caso de que el fichero del test no se haya terminado de leer, lo que se obtendrá será un salto de página, por lo que se ejecutarán los comandos incluidos dentro de la sentencia if. Lo primero que se hace dentro del condicional es obtener el valor de la variable “respuesta”, que depende del radiobutton seleccionado. Se compara este valor con el de la variable “respuestaCorrecta”. En caso de que coincida se añade a la lista “punt” los puntos obtenidos por respuesta acertada; en caso de que esté en blanco se añade a la lista “punt” el valor “0.0”; si no se cumple ninguna de las opciones anteriores, se añade a la lista “punt” el valor negativo de los puntos perdidos por respuesta fallada. A continuación, se obtiene del fichero la siguiente pregunta y las cuatro posibles respuestas, y se insertan en su label respectivo. También se usa el método set para anular la selección anterior del Radiobutton. Se vuelve al punto anterior, donde el usuario debe seleccionar la respuesta que considera correcta y pulsar sobre “siguiente”, para que esta función vuelva a ejecutarse.

Si la lectura de la línea situada después de la respuesta correcta es una cadena vacía significa que se ha llegado al final del fichero. En ese caso, hay que evaluar si la última respuesta marcada es correcta, incorrecta o está en blanco, añadir el valor que corresponda al vector “punt” y cerrar el archivo. Se mostrará por pantalla un mensaje que indicará que el test está finalizado y que se están guardando los datos, para proceder a cerrar las pantallas de Test y cronómetro y ejecutar la función guardarDatos, incluida dentro de la función siguientePregunta, que se puede observar en la Fig. 64.

```
def siguientePregunta(punt):

    def guardarPuntuacion():
        nota = 0.0

        for i in range(0, len(punt)):
            nota += punt[i]

        #Los nombres deben ser cambiados por el profesor
        fichero_Notas_Test1 = 'Notas Test 1.txt'
        fichero_Notas_Test2 = 'Notas Test 2.txt'

        if lbSelect[0] == 0:
            try:
                archivo_Notas = open(fichero_Notas_Test1, 'r+')
                archivo_Notas.read()
            except FileNotFoundError:
                archivo_Notas = open(fichero_Notas_Test1, 'w')
        elif lbSelect[0] == 1:
            try:
                archivo_Notas = open(fichero_Notas_Test2, 'r+')
                archivo_Notas.read()
            except FileNotFoundError:
                archivo_Notas = open(fichero_Notas_Test2, 'w')

        archivo_Notas.write(nombreUsuario)
        archivo_Notas.write('\n')
        archivo_Notas.write(str(nota))
        archivo_Notas.write('\n')
```

Fig. 64: Implementación de la función guardarPuntuacion

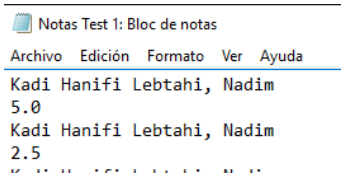


Fig. 65: Fichero "Notas Test 1.txt"

Se establece el valor de la variable "nota" en 0. Una vez hecho esto, se itera en una lista creada con el método "range()" a partir de la longitud de la lista "punt". Como en la lista "punt" estaban las puntuaciones, positivas o negativas, de cada pregunta, habrá que ir sumándolas una a una hasta dejar el valor total de la puntuación en la variable "nota".

A continuación, están los nombres de los ficheros de notas donde se guardarán las puntuaciones obtenidas. Estos nombres deben ser modificados por los profesores, igual que los nombres de los ficheros donde se encuentra el contenido de los exámenes, puesto que depende de los tests que estén activos. Así, según el test que se hubiera elegido previamente en la pantalla de exámenes, las operaciones se harán con un fichero o con otro.

Concretamente, "se intenta", mediante el método "try", abrir el fichero correspondiente, en modo "r+". Este modo es, en principio, para leer un fichero, pero el signo "+" que se incluye sirve para que también se pueda escribir. Una vez abierto se ejecuta el método read(): este método no sólo lee el contenido del fichero, como su nombre indica, sino que también vuelve a sobreescribir sobre el mismo todo lo que ha leído, evitando que la información previa del fichero se pierda. En caso de que no se pueda leer el fichero porque no existe saltará la excepción "FileNotFoundError". Si esto sucede, simplemente se abre un nuevo fichero en modo de escritura.

En el fichero, ya sea nuevo o no, se escribirá lo que muestran las líneas inferiores: El nombre del alumno, seguido de un salto de línea, la nota obtenida y, finalmente, otro salto de línea (Fig. 65). La información de los alumnos y la nota obtenida por los mismos podrá extraerse por los profesores desde el fichero.

Con esto, damos por finalizada la explicación del código del programa. Se recuerda, una vez más que se puede consultar todo el código completo en el anexo correspondiente.

4. Manual de usuario

En la explicación del código, queda bastante claro el funcionamiento del terminal interactivo y las funcionalidades que ofrece. Aun así, se ha querido hacer una breve guía de usuario para que no haya lugar a dudas.

Al abrir el programa, se entra en la pantalla principal (Fig. 66). Aquí, el alumno debe introducir su DNI (sin letra) y su clave de acceso a la UPV – todo tal y como acceda a la intranet. Después de unos segundos, si los datos ingresados son correctos saltará una ventana que indicará que los datos ingresados son correctos (Fig. 68) y se podrá acceder a la pantalla secundaria, mientras se bloquean las entradas de texto del DNI y de la clave. Si no, otra ventana indicará que los datos son incorrectos (Fig. 69) y que se pueden introducir de nuevo.

Fig. 66: Pantalla principal

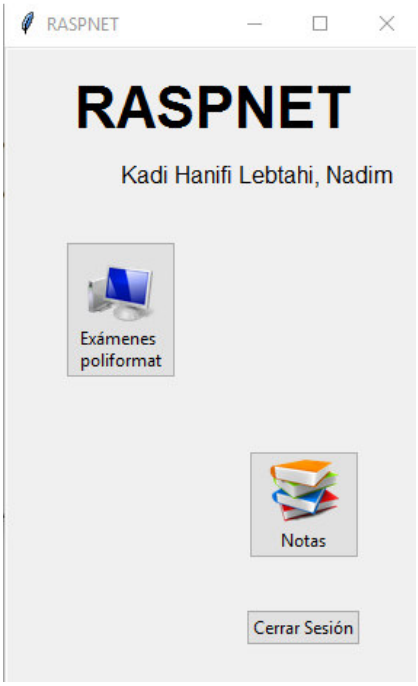


Fig. 67: Pantalla secundaria

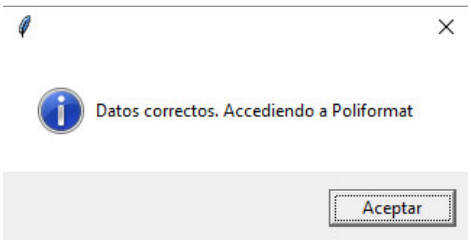


Fig. 68: Ventana ingreso correcto

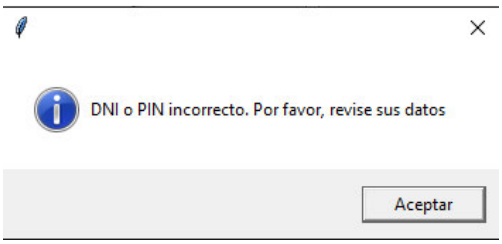


Fig. 69: Ventana ingreso incorrecto

Una vez en la pantalla secundaria (Fig. 67), si se pulsa sobre “Cerrar Sesión” saldremos de esta pantalla y se podrá volver a ingresar nuevos datos de ingreso en la pantalla principal. Aparte de este botón, hay otros dos en esta pantalla. Si se hace click sobre el botón “Notas”, se accederá a la pantalla de notas (Fig. 70). En esta, se puede ver una lista con las actas de toda la carrera. Hay que seleccionar una asignatura y confirmar pulsando sobre el botón “Mostrar Nota”, sobre

la tecla “Enter” o haciendo doble click. Esto mostrará en la parte inferior de la pantalla la nota de dicha asignatura.

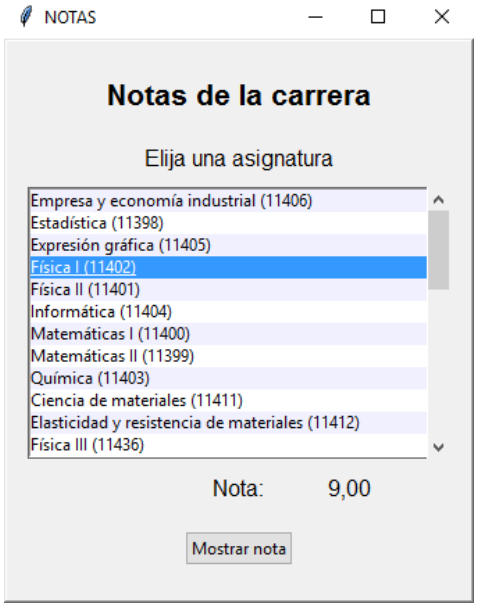


Fig. 70: Pantalla de notas

Si se pulsa sobre el botón “Exámenes Poliformat”, se abrirá una ventana con la lista de los test disponibles (Fig. 71). Se debe seleccionar uno y confirmar pulsando sobre el botón “Empezar Test”, sobre la tecla “Enter” o haciendo doble click. Al hacerlo se abrirá el test correspondiente (Fig. 72). Se debe seleccionar la respuesta que se considere correcta – o dejarlo en blanco – y pulsar sobre el botón “siguiente” para pasar a la próxima pregunta. Se vuelve a seleccionar una respuesta, se pulsa sobre “siguiente” y así sucesivamente hasta que queden respondidas a todas las preguntas o acabe el tiempo. El funcionamiento es igual al de los exámenes que se hacen en Poliformat. Señalar que, como se puede ver en la Fig. 72, hay un cronómetro que indica el tiempo restante hasta la finalización del test.

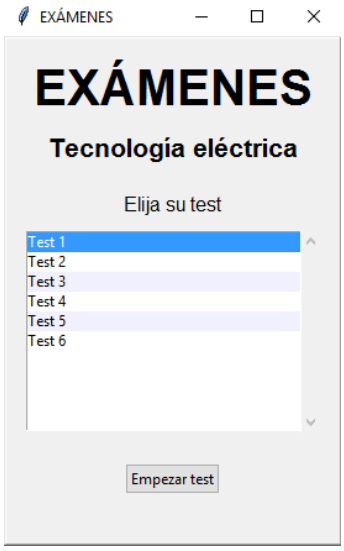


Fig. 71: Pantalla de exámenes

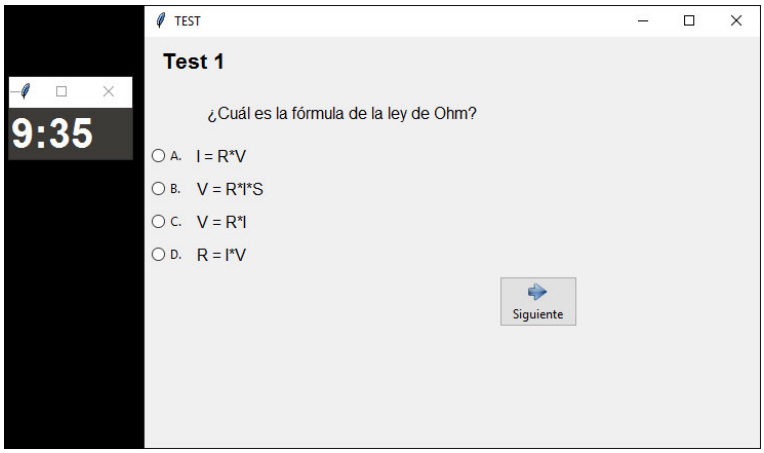


Fig. 72: Pantalla de test

Si se responde a todas las preguntas, saltará un mensaje diciendo que se ha acabado el test (Fig. 73). Cuando el alumno acepte el mensaje, se cerrarán la ventana del test y el cronómetro y se guardará la puntuación. En cambio, si se acaba el tiempo saltará otro mensaje anunciándolo (Fig. 74). Cuando se acepte, se cerrarán la ventana de test y el cronómetro, pero no se guardará la puntuación.

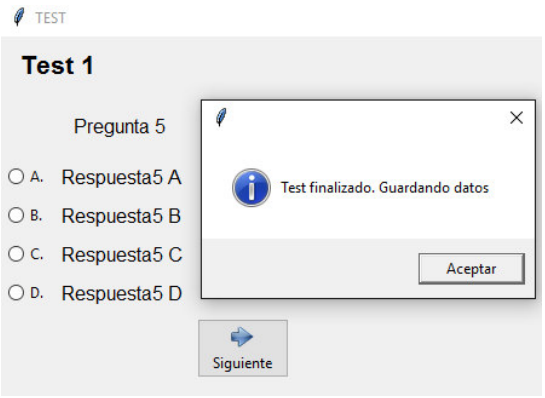


Fig. 73: Ventana test finalizado

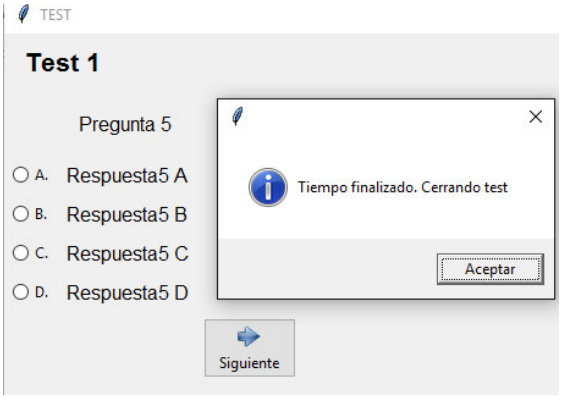


Fig. 74: Ventana tiempo finalizado

Una vez hecha cualquiera de estas operaciones, siempre se puede volver a la pantalla secundaria para hacer otra nueva operación o cerrar la sesión.

5. La Raspberry Pi en el laboratorio de Ingeniería Eléctrica

Como ya se ha comentado, la motivación de este TFG es crear un terminal que pueda acceder a UPVNET para realizar los test de la asignatura después de las prácticas. Ya se ha explicado en las páginas anteriores la implementación gráfica y funcional de este terminal.

Sin embargo, se considera que la Raspberry Pi ofrece muchas otras posibilidades para usarla en el laboratorio, y que no habría que limitarse a emplearla para la función que se ha realizado en este trabajo. Evidentemente, esto escapa al objetivo del TFG y la duración que tendría excedería la planeada. Aun así, se ha creído interesante introducir algunas de estas posibilidades para que puedan servir de base de cara a futuros proyectos.

Para empezar a hablar de esto, es clave tener en cuenta la existencia de un total de 40 pins GPIO (*General Purpose Input/Output*) (Fig. 75), de los que ya se ha hablado brevemente, mejorando a antiguas versiones de la Raspberry Pi que sólo contaban con 26. Estos pins son entradas y salidas digitales, que permitirán al usuario interactuar con la placa desde el exterior.

Fig. 75: Pins GPIO (1)

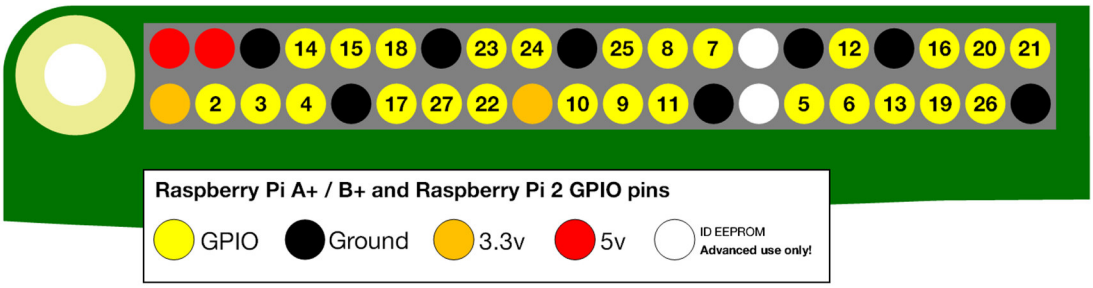


Fig. 76: Pins GPIO (2)

En la Fig. 76 se pueden apreciar los pins. De los 40 pins, 26 son GPIO (amarillos). El resto son de alimentación (ámbar y rojo) y de conexión a tierra (negro), además de los dos pins EEPROM (blanco).

En la propia página de Raspberry Pi (https://www.raspberrypi.org/learning/python-quick-reaction-game/worksheet/) está disponible un programa para empezar a aprender a utilizar los

pins de entrada y salida de la Raspberry Pi de la forma más simple: usando botones y LEDs. En la Fig. 77 se puede ver el montaje y en la Fig. 78, el código, escrito en Python.

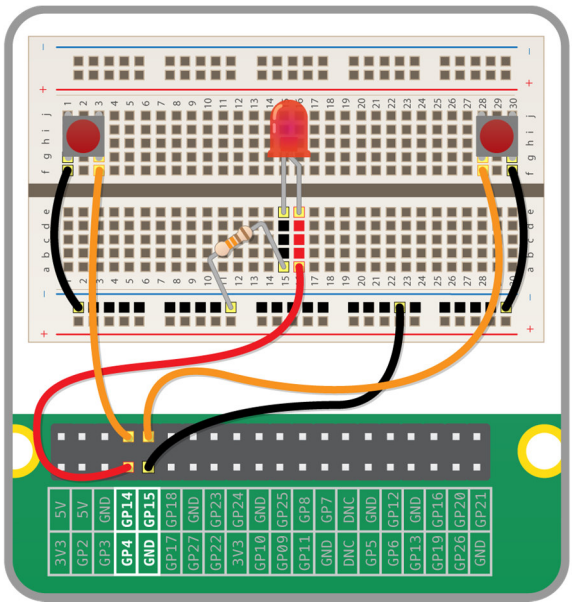


Fig. 77: Conexión pins GPIO

```python
from gpiozero import LED, Button
from time import sleep
from random import uniform

led = LED(4)
right_button = Button(15)
left_button = Button(14)

led.on()
sleep(uniform(5, 10))
led.off()

def pressed(button):
    print(button.pin.number + ' won the game')

right_button.when_pressed = pressed
left_button.when_pressed = pressed
```

Fig. 78: Conexión pins GPIO. Código Python

Como se puede ver, los dos botones están conectados a los GPIO 14 y 15, mientras que el LED está conectado a la GPIO 4. Gracias a la librería “gpiozero” se pueden referenciar las variables “led”, “right_button” y “left_button”. El juego, que requiere dos jugadores, consiste en un LED que se encenderá, mantendrá este estado entre 5 y 10 segundos y se volverá a apagar. Cuando ocurra esto, los dos jugadores deberán pulsar su botón correspondiente, y el primero en hacerlo ganará. Este programa permite combinar el uso de los pins como output, puesto que se enciende el LED desde la Raspberry Pi, con el uso de los pins como input, puesto que al pulsar sobre un botón se manda el estado a la Raspberry Pi.

En cuanto al uso que se le podría darle a estos pins en el departamento de ingeniería eléctrica, se ha pensado en que una posible aplicación sería el control de contactores desde la placa. Estos presentan únicamente dos estados: abierto o cerrado. Así pues, se podría crear una interfaz con Python que, tras conectar los contactores a los pins GPIO, permitiera acceder a ellos, según los objetivos de la práctica que se esté realizando.

Se podría añadir a esa interfaz un programa que, tras tomar las medidas necesarias en el laboratorio – voltaje, intensidad… – pudiera hacer los cálculos necesarios de forma automática, mostrar tablas o gráficas, etc. También se podría hacer preguntas, ya sea tipo test o de teclear respuesta, a medida que se va desarrollando la práctica, y en caso de no acertar se debería volver a empezar la parte de la práctica en cuestión. En resumen, sería como una interfaz que guiaría al alumno en el desarrollo de la práctica, ayudaría a obtener los datos y le permitiría interactuar con el material del laboratorio.

A diferencia de otras placas relativamente similares como el Arduino, la Raspberry Pi no cuenta con entradas y salidas analógicas, ni tampoco tiene un conversor Analógico-Digital (y viceversa). Sin embargo, con ayuda de los pins GPIO, se le puede conectar un conversor, como por ejemplo

el MCP3008, gracias al cual se podría trabajar con variables analógicas. Gracias a esto, nuevos aparatos como sensores o potenciómetros podrían ser conectados a la Raspberry Pi, e incluso se podría estudiar la inclusión de aparatos de medida como multímetros. Así, se podrían pasar las medidas analógicas de voltaje o intensidad obtenidas en el laboratorio directamente a la placa, para poder hacer operaciones, gráficas, etc. con ellas. Otra posibilidad sería regular la intensidad de la luz, en las prácticas de alumbrado.

Lo que se ha expuesto en los últimos párrafos son sólo algunos ejemplos de aplicaciones que puede tener la Raspberry Pi en un laboratorio de ingeniería eléctrica. Lamentablemente, no se puede profundizar mucho más en ellas, pero se ha pensado que resultaría interesante hacer una pequeña introducción a otros usos de la placa, que se podrían desarrollar en futuros proyectos, para no limitarnos únicamente a explicar el desarrollo del terminal con acceso a UPVNET, objetivo principal de este TFG.

6. Conclusiones

Para concluir con la memoria del TFG, se ha querido hacer un balance del trabajo realizado, comentar el cumplimiento de los objetivos y los resultados obtenidos, y hacer una valoración personal del proyecto.

En lo que respecta al desarrollo de la aplicación, se han dedicado las primeras semanas del TFG para familiarizarse con el lenguaje Python y las librerías empleadas. Aunque, como se dice, la práctica hace al maestro, antes de empezar se han consultado varios tutoriales del lenguaje y de la librería Tkinter para evitar empezar a ciegas. A medida que se ha ido programando, como suele ser habitual, han ido surgiendo más dudas; muchas veces la resolución de un problema daba lugar a uno nuevo. Por eso, el aprendizaje ha sido constante a lo largo del proyecto. Se ha consultado diversas fuentes para lograr resolver cada inconveniente, que van desde documentos oficiales que explican las características del lenguaje o de una librería hasta dudas similares planteadas por otros usuarios. Una vez terminado de escribir todo el código, se ha ordenado un poco para que quede lo más legible posible.

En cuanto a la redacción, se ha querido introducir como es debido conceptos sobre el Hardware y el Software utilizado, para que quede claro en todo momento con qué se ha trabajado. Una vez hecho esto, todo el código se ha explicado de la forma más clara posible, para que sea comprensible para cualquier lector, y se ha completado con la inclusión de una guía para los usuarios. Se ha tratado que el trabajo sea lo más claro y completo posible.

El objetivo del proyecto era desarrollar, usando la Raspberry Pi y el lenguaje Python, un terminal que permita a los alumnos acceder a UPVNET desde el laboratorio de Ingeniería Eléctrica y hacer los test de la asignatura correspondiente. Se considera que este objetivo se ha superado, ya que la aplicación accede correctamente a la Intranet de la UPV y es capaz de extraer información de ahí para reflejarla a continuación en el terminal (concretamente, el nombre de usuario y las notas obtenidas). El único pero que le podríamos poner es que, aunque se ha podido acceder a otros datos de la Intranet y de Poliformat, no se ha podido acceder a los test directamente desde UPVNET. Aun así, se ha podido solventar este inconveniente haciendo que el test al que se quiere acceder se encuentre en un fichero incluido en la Raspberry Pi, y a partir de él se pueda imprimir en pantalla todas las preguntas, permitir que el alumno responda y, finalmente guardar su nota.

Se considera que este terminal sigue estando en fase de prueba, en tanto que se pueden introducir mejoras. Por ahora, la interfaz de los test es bastante simple, incluyendo solamente preguntas tipo test con cuatro posibles respuestas. Por ejemplo, no se ha tenido en cuenta la posibilidad de introducir imágenes que acompañen una pregunta, ni tampoco la posibilidad de preguntas en las cuales el usuario debe responder manualmente. También sería interesante introducir la posibilidad de que el usuario pueda volver a ver las preguntas anteriores y cambiar su respuesta. Otra cosa que se ha planteado, dado que las notas de los alumnos se guardan en cada Raspberry Pi individualmente, es el poder acceder desde un terminal central (el ordenador del profesor, por ejemplo) a todos los ficheros donde se guardan las notas y poder extraerlas

tranquilamente. En definitiva, el objetivo ha sido cumplido, pero en caso de haber dispuesto de un poco más de tiempo se habría podido introducir ciertas mejoras.

Este TFG abre la puerta a la posibilidad de desarrollar nuevos proyectos. Los conocimientos adquiridos para acceder a la Intranet se pueden aprovechar para, en el futuro, crear nuevas aplicaciones o ampliar esta para implementar nuevas funcionalidades que necesiten acceder a la página web de la universidad. También se abre la puerta a idear nuevas aplicaciones que se podrían usar en un laboratorio de Ingeniería Eléctrica, algunas de las cuales ya han sido mencionadas en el apartado 5.

A título personal, me alegro de haber elegido este TFG. Considero que las aspiraciones que tenía inicialmente, que eran trabajar en un proyecto que me resulte interesante y aprender algo nuevo, han sido cumplidas satisfactoriamente. El hecho de trabajar con una Raspberry Pi me ha hecho ver las posibilidades que ofrece esta placa, y me anima a seguir utilizándola y desarrollar nuevas aplicaciones. También me ha gustado iniciarme en el aprendizaje del lenguaje Python, un lenguaje que, como he podido constatar, es muy potente y permite hacer una considerable cantidad de cosas.

Considero que el desarrollo del proyecto se ha desarrollado de forma bastante constante. A pesar de que a veces me he bloqueado en la programación, pensando y buscando una solución he podido resolver todos estos inconvenientes. Los tiempos se han medido bien, empezando por la fase de aprendizaje, seguida de los primeros pasos de la programación, que se ha ido superponiendo con la redacción de las primeras fases de la memoria para evitar una acumulación de trabajo, para finalmente destinar las últimas semanas a hacer los últimos retoques al programa y a terminar de redactar.

En resumen, ha sido un proyecto interesante, gracias al cual he aprendido cosas, en el cual hemos logrado el objetivo y el cual agradezco la oportunidad de haber podido desarrollar.

7. Bibliografía

[1] Raspberry Pi (s.f.). Obtenido de https://www.raspberrypi.org/

[2] Mario Pérez Esteso (2013, marzo 19). Tutorial Raspberry Pi – 1. El primer encendido. Obtenido de https://geekytheory.com/tutorial-raspberry-pi-1-el-primer-encendido/

[3] Dablarui (2013, diciembre 18). Raspberry Pi. Obtenido de http://histinf.blogs.upv.es/2013/12/18/raspberry-pi/

[4] Mario G. Bejarano M. (2013). Tutorial básicos. Obtenido de http://www.frambuesapi.co/category/tutoriales/tutorial-basico/

[5] Mario G. Bejarano M. (2013). Tutorial básicos. Obtenido de http://www.frambuesapi.co/category/tutoriales/tutorial-intermedio/

[6] Wikipedia (s.f.). Varios artículos. Obtenido de https://www.wikipedia.org/

[7] Nico Varonas (2012, Mayo 24). Los mini ordenadores SBC y su futuro. Obtenido de http://www.neoteo.com/los-mini-ordenadores-sbc-y-su-futuro

[8] Adaptaciones a otras arquitecturas (s.f.). Obtenido de https://www.debian.org/ports/

[9] Raspbian (s.f.). Obtenido de https://www.raspbian.org/

[10] Fernando R. Cabello (2014, julio 24). 9 usos prácticos de la Raspberry Pi. Obtenido de http://movilforum.com/9-usos-practicos-de-la-raspberry-pi/

[11] Marcela Díaz (2008, noviembre 23). Unix: características generales. Obtenido de http://sisopeunix.blogspot.com.es/2008/11/caracteristicas-generales.html

[12] Enrique Zetina (2012, enero 12). Características básicas y componentes del SO Unix. Obtenido de https://ezetina.wordpress.com/2010/01/12/caracteristicasunix/

[13] Richard Stallman (s.f.). ¿Qué es el software libre? Obtenido de https://www.gnu.org/philosophy/free-sw.html

[14] Richard Stallman (2004). Free software, free society

[15] Kernel (s.f.). Obtenido de http://www.ecured.cu/Kernel

[16] Eduardo Díaz (2012, noviembre 11). ¿Qué es la programación funcional? Obtenido de http://www.programando.org/blog/2012/11/que-es-la-programacion-funcional/

[17] Características generales de Debian GNU/Linux. Obtenido de https://sites.google.com/site/sogrupo15/historia-de-debiangnu-linux

[18] Víctor Manuel Sánchez (2013, marzo 13). Introducción a Python, instalación y hola mundo. Obtenido de https://geekytheory.com/introduccion-a-python-instalacion-y-hola-mundo/

[19] Deividcoptero (2014). Cursos de Python en español – Ideal para principiantes. Obtenido de https://www.youtube.com/playlist?list=PL6hPvfzEEMDZT-LXdvXpalL7WGZh3JURR

[20] Tim Peters (2004). The Zen of Python.

[21] Tutorials Point (s.f.). Obtenido de http://www.tutorialspoint.com/python/

[22] Python (s.f.). Obtenido de https://www.python.org/

[23] Asociación Python España (s.f.). Obtenido de http://www.es.python.org/

[24] IDLE (s.f.). Obtenido de https://docs.python.org/2/library/idle.html

[25] Guido van Rossum (2012, abril 9). What's new in Python 3.0. Obtenido de https://docs.python.org/3.1/whatsnew/3.0.html

[26] Moving from Python 2 to Python 3 (s.f.). Obtenido de http://ptgmedia.pearsoncmg.com/imprint_downloads/informit/promotions/python/python2python3.pdf

[27] Kenneth Reitz (s.f.). Requests: HTTP for humans. Obtenido de http://docs.python-requests.org/en/master/

[28] Mark Roseman (s.f.). TkDocs Tutorial. Obtenido de http://www.tkdocs.com/tutorial/index.html

[29] R. Ors, A. Martí, A. Pérez, J. Gracia, L.J. Saiz (2012). Apuntes de Informática Industrial. Ed. UPV, Ref. 678.

[30] I. García, J.A. Gil (2016). Apuntes de Diseño de Aplicaciones para Dispositivos Móviles. UPV.

PRESUPUESTO

ÍNDICE

Aquí, se va a desglosar el coste económico que supondría el desarrollo de este proyecto. Se trata de un proyecto a menor escala comparado con lo que podría ser, por ejemplo, el diseño de una instalación, por lo que el número de recursos es menor y el presupuesto será mucho más simple de lo que puede ser en otros proyectos. Aun así, es importante contabilizar todos los recursos utilizados.

Se desglosarán los precios descompuestos de cada unidad de obra: la programación del terminal y el hardware. El software no se contabiliza puesto que todo el que ha sido utilizado en el proyecto es gratuito. A continuación se calcularán los presupuestos parciales para, finalmente, mostrar el presupuesto total.

1. Cuadro de precios descompuestos

1.1. Programación del terminal

En este apartado se desglosará el presupuesto necesario para la programación, concretamente, el sueldo del ingeniero encargado de realizar el proyecto. Según la página http://www.tusalario.es/, se estima el sueldo medio en España de un ingeniero recién titulado sin experiencia laboral en 2178€/mes brutos. Suponiendo 40 horas de trabajo semanales, y teniendo en cuenta que un mes tiene 4 semanas, el coste por hora de trabajo de un ingeniero ascendería a 13,61€/h. Con esto, estimando la duración de cada fase del trabajo, se obtienen los costes directos de esta unidad de obra. También se ha tenido en cuenta un porcentaje de gastos directos complementarios y un porcentaje de gastos indirectos.

UO 1.1	Programación del terminal				
Código		Descripción	Precio	Rendim.	Importe
MO.GITI	h	GITI – Planteamiento del problema	13,61	3	40,84
MO.GITI	h	GITI – Reuniones	13,61	7	95,29
MO.GITI	h	GITI – Aprendizaje	13,61	100	1361
MO.GITI	h	GITI – Desarrollo del proyecto	13,61	100	1361
MO.GITI	h	GITI – Redacción de la memoria	13,61	90	1225,12
	%	Costes directos complementarios		0,01	40,83
	%	Costes indirectos		0,02	81,66
Total					**4205,74**

1.2. Hardware

Aquí se desglosarán todos los materiales necesarios para implementar un terminal Raspnet. Los materiales de la tabla no son exactamente los mismos que se han utilizado para el desarrollo del proyecto, sino que se ha pensado en su objetivo final, que no es otro que implementarlo en el laboratorio de ingeniería eléctrica. Por ello, se emplea un hardware que resulta más interesante para una implementación a mayor escala, como puede ser el Dongle WiFi (habiendo varias placas, no resultaría interesante conectarlas todas a la red Ethernet) o la pantalla táctil, que nos evita conectar pantalla, teclado y ratón, lo que supone un ahorro considerable de espacio y dinero.

Recordar que los enlaces a páginas de compra y las fichas técnicas de todos estos materiales se pueden encontrar en el anexo.

UO 1.2	Hardware				
Código		**Descripción**	**Precio**	**Rendim.**	**Importe**
MT.RPI	ud	Raspberry Pi 2 Modelo B	31,6	1	31,6
MT.FTAL	ud	Fuente de alimentación	0[1]	1	0
MT.SD	ud	Tarjeta SD SanDisk Ultra 16 GB	5,9	1	5,9
MT.ENC	ud	Raspberry Pi Official Enclosure (caja protectora)	8,09	1	8,09
MT.TS	ud	7” Touchscreen Display for Raspberry Pi	72,19[2]	1	72,19
MT.WIPI	ud	Wifi USB Dongle for Raspberry Pi	17,08	1	17,08
	%	Costes directos complementarios		0,01	1,35
	%	Costes indirectos		0,02	2,70
Total					**138,91**

2. Presupuesto parcial

Para el presupuesto se ha supuesto la implementación de 15 terminales en el laboratorio de Ingeniería Eléctrica.

Unidad de obra	Cantidad	Precio	Coste total
UO 1.1: Programación del terminal	1	4205,74	4205,74
UO 1.2: Hardware	15	138,91	2083,65
Total			**6289,39**

3. Presupuesto total

PRESUPUESTO DE EJECUCIÓN MATERIAL	**6289,39**
Gastos generales (13%)	817,62
Beneficio industrial (6%)	377,36
PRESUPUESTO DE INVERSIÓN	**7484,37**
IVA (21%)	1571,72
PRESUPUESTO BASE DE LICITACIÓN	**9056,09**

Asciende el presupuesto a la expresada cantidad de:

NUEVE MIL CINCUENTA Y SEIS EUROS CON NUEVE CÉNTIMOS

4. Valoración económica

Contabilizando tanto el sueldo que correspondería al ingeniero como el precio de los componentes, pensando en una implementación para 15 terminales, vemos claramente que el precio supondría un ahorro considerable con respecto al uso de ordenadores, sin mencionar la comodidad y el ahorro en espacio. Si se quisiera ampliar las funciones de las que dispone el terminal, sería asumible el coste del ingeniero encargado de hacerlo, y se podría amortizar fácilmente al cabo de un tiempo.

En definitiva, el presupuesto nos permite confirmar lo que ya sospechábamos: resulta económicamente rentable emplear Raspberry Pi en el laboratorio para hacer los test de las

prácticas, con respecto al precio de los equipos utilizados actualmente. Resultaría todavía más rentable si se decidiera darle otros usos, puesto que el precio del ingeniero es asumible.

---

[1] El cargador viene incluido con la Raspberry Pi.

[2] Descuento del 10% al adquirir más de 10 unidades.

PLANOS

Para completar el TFG hemos hecho unos planos con las medidas de la Raspberry Pi. En esta página (Fig. 79) se puede ver el modelado de la pieza con Inventor, mientras que en la siguiente página tenemos el plano.

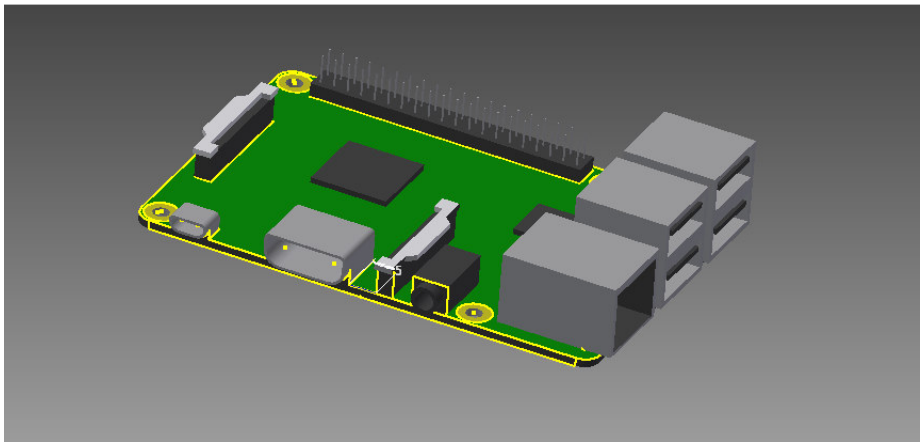
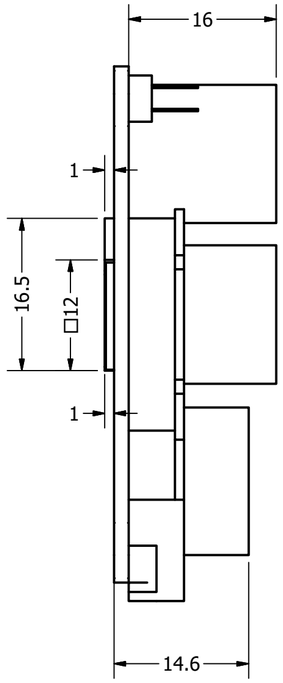
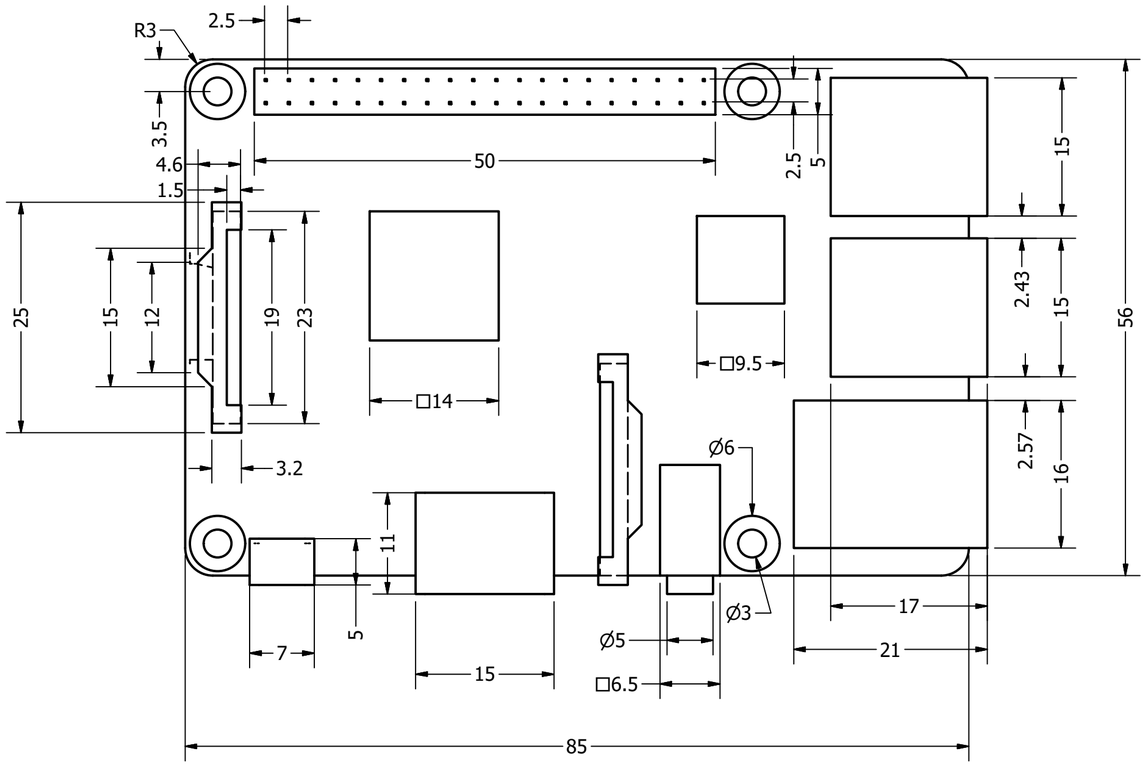
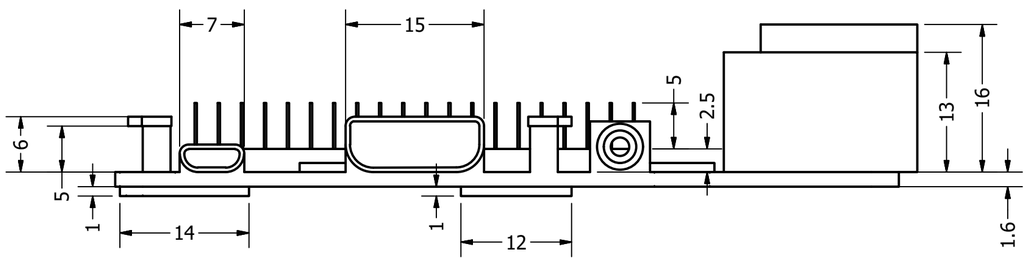


Fig. 79: Raspberry Pi con Autodesk Inventor







TRABAJO DE FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES UNIVERSITAT POLITÈCNICA DE VALÈNCIA · ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA	DISEÑO DE TERMINAL PARA CONEXIÓN A UPVNET DESDE EL LABORATORIO DE INGENIERÍA ELÉCTRICA A UPVNET BASADO EN RASPBERRY PI	Raspberry Pi	Julio 2016	1
		Nadim Kadi Hanifi Lebtahi	2:1	

ANEXOS

ÍNDICE

1. Fichas técnicas

Raspberry Pi 2 Modelo B

http://www.alliedelec.com/raspberry-pi-raspberry-pi-2-model-b/70465426/

Tarjeta micro SD SanDisk Ultra - 16 GB

https://www.amazon.es/SanDisk-SDSQUNC-016G-GZFMA-Android-microSDHC-adaptador/dp/B013UDL5V6/279-7447474-6514012?ie=UTF8&redirect=true&tag=masmanuti-21

Televisor Saivod modelo LCD 632CI-E

http://www.ciao.es/Saivod_LCD_632_CI__704783

Ratón Logitech Mouse M150

http://www.logitech.com/es-es/product/mouse-m150

Teclado Logitech K120

http://www.logitech.com/es-es/product/k120

Raspberry Pi Micro USB Power Suply

http://www.alliedelec.com/raspberry-pi-pi-power-supply-white/70812912/

Procesador ARM de la Raspberry Pi

http://infocenter.arm.com/help/topic/com.arm.doc.ddi0301h/DDI0301H_arm1176jzfs_r0p7_trm.pdf

Raspberry Pi official case (caja protectora)

https://www.adafruit.com/product/2604

Pi Foundation Display - 7" Touchscreen Display for Raspberry Pi

https://www.adafruit.com/products/2718

WiFi USB Dongle for Raspberry Pi

https://www.element14.com/community/docs/DOC-69361/l/wifi-usb-dongle-for-raspberry-pi

2. Código Python

```python
from tkinter import *                           #Importamos las librerías gráficas
from tkinter import ttk, font, messagebox
from PIL import ImageTk, Image

import requests                                 #Importamos las librerías de internet
import http.cookiejar
from bs4 import BeautifulSoup
import cronometro

def guardarDatos(*args):
    with requests.Session() as s:

        jar = http.cookiejar.CookieJar()

        #Links URL
        login_url  = 'https://intranet.upv.es/exp/aute_intranet'
        inicio_url = 'https://intranet.upv.es/pls/soalu/sic_menu.MiUPV?P_IDIOMA=c&P_MODO=alumno'
        nombre_url = 'https://intranet.upv.es/pls/soalu/sic_menu.Alumno?P_IDIOMA=c'
        notas_url  = 'https://intranet.upv.es/pls/soalu/sic_asi.notes_temaalu_asi'

        datos_login = {
                       'dni': dni.get(),
                       'clau': pin.get()
                       }

        datos_notas = {'p_curso' : '1'}

        s.get(login_url, cookies=jar)
        s.post(login_url, data=datos_login, cookies=jar)

        entDni.delete(0, 'end')
        entPin.delete(0, 'end')

        req = s.get(inicio_url)

        if req.headers['Content-Length']=='11858':
            messagebox.showinfo(message='DNI o PIN incorrecto. Por favor, revise sus datos', parent=mainframe)
            entDni.focus()

        else:
            buscNombre = s.get(nombre_url)
            htmlNombre = BeautifulSoup(buscNombre.text, 'html.parser')
            nombre = htmlNombre.find('a', {'name' : 'panel_155'}).getText()

            notasUPV = s.get(notas_url, cookies=jar)
            notasUPV = s.post(notas_url, cookies=jar, data=datos_notas)
            htmlNotas = BeautifulSoup(notasUPV.text, 'html.parser')

            listAsignaturas = []

            table = htmlNotas.find('table', {'class' : 'upv_listacolumnas'})

            for asignatura in table.find_all('td', {'class' : 'upv_listacolumnassubtitulo'}):
                listAsignaturas.append(asignatura.getText())

            listDatos = []
            listNotas = []
            rango     = range(len(listAsignaturas))
            iterAsig  = list(rango)

            for datosAsignatura in table.find_all('tr', {'class' : 'upv_listanon'}):
                for nota in datosAsignatura.find_all('td'):
                    listDatos.append(nota.getText())

            for i in iterAsig:
                listNotas.append(listDatos[2+6*i])

            messagebox.showinfo(message='Datos correctos. Accediendo a Poliformat', parent=mainframe)
            entDni.config(state='disabled')
            entPin.config(state='disabled')
            crearVentanaSecundaria(nombre, listAsignaturas, listNotas)

def crearVentanaSecundaria(nombreUsuario, lAsignaturas, listaNotas):

    def cerrarSesion():
        wdw2.destroy()
        entDni.config(state='enabled')
        entPin.config(state='enabled')
        entDni.focus()

    def crearVentanaNotas(lAsignaturas, listaNotas):

        def mostrarNota (*args):
            lbSeleccion = lbAsignaturas.curselection()
            numAsig = int(lbSeleccion[0])
            notaAsig = listaNotas[numAsig]
            nota.set(notaAsig)

        wdwNotas = Toplevel()
```

```
90          wdwNotas.title('NOTAS')
91
92          wdwNotas.grab_set()
93
94          notasframe = ttk.Frame(wdwNotas, padding=15, borderwidth=2, relief='raised')
95          notasframe.grid(column=0, row=0, sticky=(N, W, E, S))
96
97          lblTituloNotas = ttk.Label(notasframe, text='Notas de la carrera', font='Helvetica 16 bold')
98          lblElegir = ttk.Label(notasframe, text='Elija una asignatura', font='Helvetica 12')
99
100         listaAsignaturas = StringVar(value=lAsignaturas)
101         lbAsignaturas = Listbox(notasframe, height=12, width=50, listvariable=listaAsignaturas)
102
103         for i in range(0,len(lAsignaturas),2):
104                 lbAsignaturas.itemconfigure(i, background='#f0f0ff')
105
106         sbAsignaturas = ttk.Scrollbar(notasframe, orient=VERTICAL, command=lbAsignaturas.yview)
107         lbAsignaturas.configure(yscrollcommand=sbAsignaturas.set)
108         lbAsignaturas.selection_set(0)
109
110         lblNotaObt = ttk.Label(notasframe, text='Nota:', font='Helvetica 12')
111
112         nota = StringVar()
113         lblNota = ttk.Label(notasframe, textvariable=nota, font='Helvetica 12')
114
115         btMostrar = ttk.Button(notasframe, text='Mostrar nota', command=mostrarNota)
116
117         lblTituloNotas.grid(column=0, row=0, sticky=(N), pady=10)
118         lblElegir.grid(column=0, row=1, pady=10)
119         lbAsignaturas.grid(column=0, row=2, sticky=(N, S, E, W))
120         sbAsignaturas.grid(column=0, row=2, sticky=(N, S, E))
121         lblNotaObt.grid(column=0, row=3, sticky=S, pady=10)
122         lblNota.grid(column=0, row=3, sticky=E, padx=55, pady=10)
123         btMostrar.grid(column=0, row=4, padx=10, pady=10)
124
125         wdwNotas.bind('<Return>', mostrarNota)
126         lbAsignaturas.bind('<Double-1>', mostrarNota)
127
128     def crearVentanaExamenes ():
129
130         def abrirTest (*args):
131
132             def siguientePregunta(punt):
133
134                 def guardarPuntuacion():
135                     nota = 0.0
136
137                     for i in range(0, len(punt)):
138                         nota += punt[i]
139
140                     #Los nombres deben ser cambiados por el profesor
141                     fichero_Notas_Test1 = 'Notas Test 1.txt'
142                     fichero_Notas_Test2 = 'Notas Test 2.txt'
143
144                     if lbSelect[0] == 0:
145                         try:
146                             archivo_Notas = open(fichero_Notas_Test1, 'r+')
147                             archivo_Notas.read()
148                         except FileNotFoundError:
149                             archivo_Notas = open(fichero_Notas_Test1, 'w')
150                     elif lbSelect[0] == 1:
151                         try:
152                             archivo_Notas = open(fichero_Notas_Test2, 'r+')
153                             archivo_Notas.read()
154                         except FileNotFoundError:
155                             archivo_Notas = open(fichero_Notas_Test2, 'w')
156
157                     archivo_Notas.write(nombreUsuario)
158                     archivo_Notas.write('\n')
159                     archivo_Notas.write(str(nota))
160                     archivo_Notas.write('\n')
161
162                 respuestaCorrecta = archivo_Test.readline().strip()
163
164                 if archivo_Test.readline() != '':
165
166                     if respuesta.get() == respuestaCorrecta:
167                         punt.append(ptsxPregunta)
168                     elif respuesta.get() == '':
169                         punt.append(0.0)
170                     else:
171                         punt.append(-ptsxFallo)
172
173                     pregunta = archivo_Test.readline().strip()
174                     respuestaA = archivo_Test.readline().strip()
175                     respuestaB = archivo_Test.readline().strip()
176                     respuestaC = archivo_Test.readline().strip()
177                     respuestaD = archivo_Test.readline().strip()
178
179                     respuesta.set('')
180
181                     lblPregunta['text'] = pregunta
182                     lblRespA['text'] = respuestaA
183                     lblRespB['text'] = respuestaB
184                     lblRespC['text'] = respuestaC
185                     lblRespD['text'] = respuestaD
186
```

```
            else:
                if respuesta.get() == respuestaCorrecta:
                    punt.append(ptsxPregunta)
                elif respuesta.get() == '':
                    punt.append(0.0)
                else:
                    punt.append(-ptsxFallo)

                archivo_Test.close()
                messagebox.showinfo(message='Test finalizado. Guardando datos', parent=wdwTest)
                guardarPuntuacion()
                wdwCrono.destroy()
                wdwTest.destroy()

        def countdown(secs):

            if secs >= 0:
                if wdwCrono.state() == 'normal':
                    minutos = secs//60
                    segundos = secs - 60*minutos
                    tiempo = str(minutos) + ":" + str(segundos).zfill(2)

                    root.after(1000, countdown, secs-1)
                    lblClock['text'] = tiempo

            else:
                messagebox.showinfo(message='Tiempo finalizado. Cerrando test', parent=wdwTest)
                wdwTest.destroy()
                wdwCrono.destroy()

        lbSelect = lbTest.curselection()

        #Los nombres deben ser cambiados por el profesor
        fichero_Test1 = 'Test 1.txt'
        fichero_Test2 = 'Test 2.txt'

        if lbSelect[0] == 0:
            archivo_Test = open(fichero_Test1, 'r')
        elif lbSelect[0] == 1:
            archivo_Test = open(fichero_Test2, 'r')

        ptsxPregunta = float(archivo_Test.readline())
        ptsxFallo = float(archivo_Test.readline())
        nombreTest = archivo_Test.readline().strip()
        tiempoTest = int(archivo_Test.readline())

        archivo_Test.readline()

        punt = []

        pregunta = archivo_Test.readline().strip()
        respuestaA = archivo_Test.readline().strip()
        respuestaB = archivo_Test.readline().strip()
        respuestaC = archivo_Test.readline().strip()
        respuestaD = archivo_Test.readline().strip()

        wdwTest = Toplevel()
        wdwTest.title('TEST')
        wdwTest.geometry('600x400')

        wdwTest.grab_set()

        lblNombreTest = ttk.Label(wdwTest, text=nombreTest, font='Helvetica 16 bold')
        lblPregunta = ttk.Label(wdwTest, text=pregunta, font='Helvetica 12')
        lblRespA = ttk.Label(wdwTest, text=respuestaA, font='Helvetica 12')
        lblRespB = ttk.Label(wdwTest, text=respuestaB, font='Helvetica 12')
        lblRespC = ttk.Label(wdwTest, text=respuestaC, font='Helvetica 12')
        lblRespD = ttk.Label(wdwTest, text=respuestaD, font='Helvetica 12')

        respuesta = StringVar()
        rbA = ttk.Radiobutton(wdwTest, text='A.', variable=respuesta, value='A')
        rbB = ttk.Radiobutton(wdwTest, text='B.', variable=respuesta, value='B')
        rbC = ttk.Radiobutton(wdwTest, text='C.', variable=respuesta, value='C')
        rbD = ttk.Radiobutton(wdwTest, text='D.', variable=respuesta, value='D')

        imgSiguiente = Image.open('next.png')
        imgSiguiente.thumbnail((30, 20), Image.ANTIALIAS)
        imgNext = ImageTk.PhotoImage(imgSiguiente)
        btNext = ttk.Button(wdwTest, image=imgNext, text='Siguiente', compound='top',
                            command=Lambda : siguientePregunta(punt))
        btNext.image = imgNext

        lblNombreTest.grid(column=0, row=0, columnspan=2, sticky=(N, W), padx=15, pady=10)
        lblPregunta.grid(column=1, row=1, sticky=W, padx=15, pady=15)
        rbA.grid(column=0, row=2, sticky=(W, N), padx=5, pady=5)
        rbB.grid(column=0, row=3, sticky=(W, N), padx=5, pady=5)
        rbC.grid(column=0, row=4, sticky=(W, N), padx=5, pady=5)
        rbD.grid(column=0, row=5, sticky=(W, N), padx=5, pady=5)
        lblRespA.grid(column=1, row=2, sticky=(W, N), padx=5, pady=5)
        lblRespB.grid(column=1, row=3, sticky=(W, N), padx=5, pady=5)
        lblRespC.grid(column=1, row=4, sticky=(W, N), padx=5, pady=5)
        lblRespD.grid(column=1, row=5, sticky=(W, N), padx=5, pady=5)
        btNext.grid(column=2, row=6, sticky=(N, E), padx=5, pady=5)

```

```
281             wdwCrono = Toplevel()
282             wdwCrono.title('CRONÓMETRO')
283 
284             lblClock = ttk.Label(wdwCrono, font='ubuntu 30 bold', background='#3C3B37', foreground='white')
285             lblClock.pack(fill=BOTH, expand=1)
286 
287             countdown(60*tiempoTest)
288 
289             wdwExam.destroy()
290 
291         fichero_listaTest = 'Lista Test.txt'
292         archivo_listaTest = open(fichero_listaTest, 'r')
293 
294         asignaturaTest = archivo_listaTest.readline().strip()
295 
296         lTest0 = []
297         lTest = []
298 
299         for linea in archivo_listaTest:
300             lTest0.append(linea)
301 
302         for i in lTest0:
303             lTest.append(i.strip())
304 
305         archivo_listaTest.close()
306 
307         wdwExam = Toplevel()
308         wdwExam.title('EXÁMENES')
309 
310         wdwExam.grab_set()
311 
312         examframe = ttk.Frame(wdwExam, padding=15, borderwidth=2, relief='raised')
313         examframe.grid(column=0, row=0, sticky=(N, W, E, S))
314 
315         lblTituloExam = ttk.Label(examframe, text='EXÁMENES', font='Helvetica 30 bold')
316         lblAsignatura = ttk.Label(examframe, text=asignaturaTest, font='Helvetica 16 bold')
317         lblTest = ttk.Label(examframe, text='Elija su test', font='Helvetica 12')
318 
319         listaTest = StringVar(value=lTest)
320 
321         lbTest = Listbox(examframe, width=30, listvariable=listaTest)
322         sbTest = ttk.Scrollbar(examframe, orient=VERTICAL, command=lbTest.yview)
323         lbTest.configure(yscrollcommand=sbTest.set)
324         lbTest.selection_set(0)
325 
326         for i in range(0,len(lTest),2):
327             lbTest.itemconfigure(i, background='#f0f0ff')
328 
329         btAcceder = ttk.Button(examframe, text='Empezar test', command=abrirTest)
330 
331         lblTituloExam.grid(column=0, row=0, sticky=N, padx=5)
332         lblAsignatura.grid(column=0, row=1, pady=10)
333         lblTest.grid(column=0, row=2, pady=10)
334         lbTest.grid(column=0, row=3, sticky=(N, S, E, W))
335         sbTest.grid(column=0, row=3, sticky=(N, S, E))
336         btAcceder.grid(column=0, row=4, padx=25, pady=25)
337 
338         wdwExam.bind('<Return>', abrirTest)
339         lbTest.bind('<Double-1>', abrirTest)
340 
341     wdw2 = Toplevel()
342     wdw2.title('RASPNET')
343 
344     wdw2.grab_set()
345 
346     secframe = ttk.Frame(wdw2, padding=15, borderwidth=2, relief='raised')
347     secframe.grid(column=0, row=0, sticky=(N, W, E, S))
348 
349     lblTitulo2 = ttk.Label(secframe, text='RASPNET', font='Helvetica 30 bold')
350     lblNombre = ttk.Label(secframe, text=nombreUsuario, font='Helvetica 12')
351 
352     imgNot = Image.open('books.png')
353     imgNot.thumbnail((50, 50), Image.ANTIALIAS)
354     imgNotas = ImageTk.PhotoImage(imgNot)
355     btNotas = ttk.Button(secframe, text='Notas', image=imgNotas, compound='top',
356                          command=lambda : crearVentanaNotas(lAsignaturas, listaNotas))
357 
358     imgEx = Image.open('pc.png')
359     imgEx.thumbnail((50, 50), Image.ANTIALIAS)
360     imgExamenes = ImageTk.PhotoImage(imgEx)
361     btExamenes = ttk.Button(secframe, text='Exámenes\npoliformat', image=imgExamenes, compound='top',
362                          command=crearVentanaExamenes)
363 
364     btSalir = ttk.Button(secframe, text='Cerrar Sesión', command=cerrarSesion)
365 
366     lblTitulo2.grid(column=0, row=0, columnspan=2, sticky=N, padx=5)
367     lblNombre.grid(column=0, row=1, columnspan=2, sticky=E, pady=10)
368     btExamenes.grid(column=0, row=2, padx=25, pady=25)
369     btNotas.grid(column=1, row=3, padx=25, pady=25)
370     btSalir.grid(column=1, row=4, padx=10, pady=10)
371 
372     wdw2.mainloop()
373 
```

```
root = Tk()
root.title('Acceso a RASPNET')

mainframe = ttk.Frame(root, padding=15, borderwidth=2, relief='raised')
mainframe.grid(column=0, row=0, sticky=(N, W, E, S))

lblTitulo = ttk.Label(mainframe, text='ACCESO A RASPNET', font='Helvetica 30 bold')

imgLib = Image.open('libros.png')
imgLib.thumbnail((250, 250), Image.ANTIALIAS)
imgLibros = ImageTk.PhotoImage(imgLib)
lblImgLibros = ttk.Label(mainframe, image=imgLibros)

lblDatos = ttk.Label(mainframe, text='Introduzca sus datos', font='Helvetica 16')
lblDni = ttk.Label(mainframe, text='DNI', font='Helvetica 16')
lblPin = ttk.Label(mainframe, text='PIN', font='Helvetica 16')

dni = StringVar()
entDni = ttk.Entry(mainframe, textvariable=dni, show='*', font='Helvetica 10')
pin = StringVar()
entPin = ttk.Entry(mainframe, textvariable=pin, show='*', font='Helvetica 10')

btAcceso = ttk.Button(mainframe, text='Acceder', command=guardarDatos)

lblTitulo.grid(column=0, row=0, columnspan=3, sticky=N, padx=5)
lblImgLibros.grid(column=0, row=1, rowspan=6, pady=('20 5'))
lblDatos.grid(column=1, row=1, columnspan=3, sticky=(W, N), padx=20, pady=('10 10'))
lblDni.grid(column=1, row=2, sticky=(W, N), padx=20, pady=0)
lblPin.grid(column=1, row=4, sticky=(W, N), padx=20, pady=0)
entDni.grid(column=1, row=3, sticky=(W, N), padx=20, pady=0)
entPin.grid(column=1, row=5, sticky=(W, N), padx=20, pady=0)
btAcceso.grid(column=2, row=6)

#Las siguientes líneas de código permiten que la ventana se expanda
'''
root.columnconfigure(0, weight=1)
root.rowconfigure(0, weight=1)
mainframe.columnconfigure(0, weight=1)
mainframe.columnconfigure(1, weight=1)
mainframe.columnconfigure(2, weight=1)
mainframe.rowconfigure(0, weight=1)
mainframe.rowconfigure(1, weight=1)
mainframe.rowconfigure(2, weight=1)
mainframe.rowconfigure(3, weight=1)
mainframe.rowconfigure(4, weight=1)
mainframe.rowconfigure(5, weight=1)
mainframe.rowconfigure(6, weight=1)
'''

entDni.focus()
root.bind('<Return>', guardarDatos)

root.mainloop()
```