PRÁCTICAS

Ingeniería de Sistemas Software Basados en Conocimiento

Gonzalo Márquez de Torres

Curso 2022-23

ÍNDICE

PRÁCTICA I. BASES PARA LA IMPLEMENTACIÓN DE UNA ARQUITECTURA MODELO VISTAS-CONTROLADOR

PRÁCTICA II. BASES PARA LA IMPLEMENTACIÓN DE UNA ARQUITECTURA MODELO-VISTAS-CONTROLADOR.

1. PYQT.

1.1. Creación de Ventanas y Botones

Los widgets básicos se encuentran en el módulo *PyQt5.QtWidgets*. El parámetro *sys.argv* es una lista de argumentos de una línea de comando. Un dato importante es que los scripts de *Python* se pueden ejecutar desde el terminal, siendo una forma de controlar el inicio de estos.

El widget *QWidget* es la clase base de todos los objetos de interfaz de usuario en *PyQt5*. El constructor predeterminado para *QWidget* no tiene padre y es conocido como ventana. Existirán diferentes métodos para modificar diferentes parámetros de la ventana, como, por ejemplo, *resize* permite modificar el tamaño del widget, el título de la ventana se podrá cambiar con *setWindowTitle*, el cual se muestra en la barra de título de la misma ventana. Además, también podremos añadir un determinado icono (*imagen pequeña que se muestra en la esquina superior izquierda junto al título*), con el método *setWindowIcon*. Por último, el método *show* muestra el widget en la pantalla, el cual se crea primero en la memoria y luego se muestra en la pantalla. El código será accesible en http://www.uco.es/~i02matog/practica1/p1_ejemplo2_firstprograms.txt.

El bucle principal recibe eventos del sistema de ventanas y los envía a los widgets de la aplicación. El bucle principal finaliza si llamamos al método de salida o se destruye el widget principal. El método *sys.exit* asegura una salida limpia, informando al medio cómo finalizó la aplicación.

Ahora creamos una nueva clase llamada *Example*, la cual hereda de la clase *QWidget*. Esto significa que llamamos a dos constructores: el primero para la clase principal y el segundo para la clase heredada. El método super devuelve el objeto principal de la clase *Exemple* y llamamos a su constructor. El método *__init__* es un método constructor en lenguaje Python. La creación de la GUI se delega al método *initUI*.

Este programa será accesible por medio del siguiente enlace, http://www.uco.es/~i02matog/practica1/p1_ejemplo3_botones1.txt.

El método *setFont* permite modificar la fuente y el tamaño de la letra para representar información sobre herramientas. Para crear una información sobre herramientas, llamamos al método *setTooltip*.

```
class Example(QWidget):

    def __init__(self):
        super().__init__()
        self.initUI()


    def initUI(self):
        QToolTip.setFont(QFont('Calibri', 12)) # Modifica el tipo de letra y tamaño

        self.setToolTip('This is a <b>QWidget</b> widget') # Crea información sobre la herramienta

        btn = QPushButton('Cerrar', self) # Creación del boton
        btn.clicked.connect(QApplication.instance().quit)
        btn.setToolTip('This is a <b>QPushButton</b> widget') # Crea información sobre la herramienta
        btn.resize(btn.sizeHint()) # Tamaño idoneo predefinido
        btn.move(50, 50) # Determinar el tamaño de la ventana

        self.setGeometry(300, 300, 300, 200) # Geometria de la ventana
        self.setWindowTitle('Ejemplo Botones') # Titulo de la ventana
        self.show() # Mostrar ventana


def main():

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())


if __name__ == '__main__':
    main()
```

Imagen 2.1.1. Programa para crear una ventana modificando algunos parámetros.

1.2. Eventos y Señales.

Las aplicaciones GUI están dirigidas por eventos, los cuales son generados principalmente por el usuario de una aplicación. Aunque también pueden generarse por otros medios (*una conexión a Internet, un administrador de ventanas o un temporizador*). Cuando llamamos al método *exec_* de la aplicación, la aplicación ingresa al ciclo principal. El bucle principal obtiene eventos y los envía a los objetos. En el modelo de evento, hay tres participantes:

- Origen del evento: objeto cuyo estado cambia y genera eventos.
- Objeto de evento: encapsula los cambios de estado en la fuente del evento.
- Destino del evento: objeto que desea recibir una notificación.
- Objetivo del evento: delega la tarea de manejar un evento al destino del evento.

PyQt5 tiene un mecanismo único de señal y ranura para manejar eventos. Las señales y las ranuras (*cualquier objeto de Python*) se utilizan para la comunicación entre objetos, como se puede observar en el siguiente enlace del código, http://www.uco.es/~i02matog/practica1/p1_ejemplo11_senales.txt. Se emite una señal cuando ocurre un evento en particular. Se llama a una ranura cuando se emite su señal conectada.

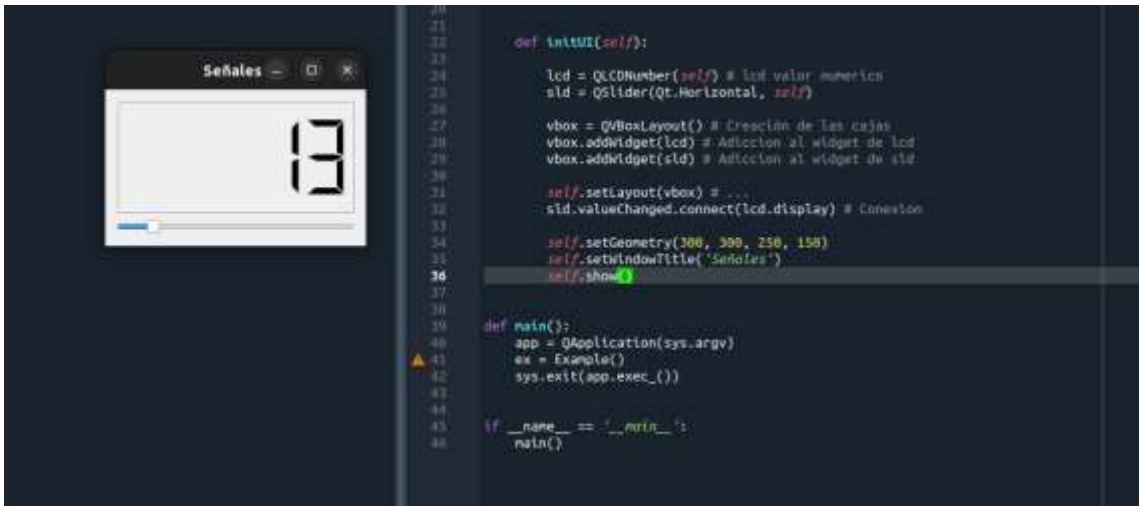

Imagen 2.1.2. Creación de un evento que modifica su valor a medida que avanza.

En el ejemplo, mostramos un *QtGui.QLCDNumber* y un *QtGui.QSlider*. Cambiamos el número de *lcd* arrastrando la pestaña deslizante. Con una señal *valueChanged* del control deslizante variará el número de lcd.

El emisor es un objeto que envía una señal y el receptor es el objeto que recibe la señal. La ranura es el método que reacciona a la señal. Los eventos en PyQt5 se procesan a menudo mediante la reimplementación de los controladores de eventos. En nuestro ejemplo, introducimos el controlador de eventos *keyPressEvent*. Si hacemos pulsamos la tecla botón Escape, la aplicación finaliza.

```
def keyPressEvent(self, e):
    if e.key() == Qt.Key_Escape:
        self.close()
```

Imagen 2.1.3. Función para que al pulsar Escape cierre la pestaña.

El objeto de evento es un objeto de *Python* que contiene una serie de atributos que describen el evento. El objeto de evento es específico del tipo de evento generado. Las coordenadas x e y se muestran en un widget *QLabel*.

El seguimiento del ratón se podrá activar con *setMouseTracking(true)*, ya que está deshabilitado de forma predeterminada. Por lo que el widget solo recibe eventos de movimiento del cuando se mueve el ratón por la ventana, incluso si no se presiona ningún botón.

El objeto de evento contiene datos sobre el evento que se desencadenó. En nuestro caso, un evento de movimiento del ratón. Con los métodos x e y determinamos las coordenadas x e y de la ventana del puntero del ratón. A veces conviene saber qué widget es el emisor de una señal. Para esto, PyQt5 tiene el método del remitente.

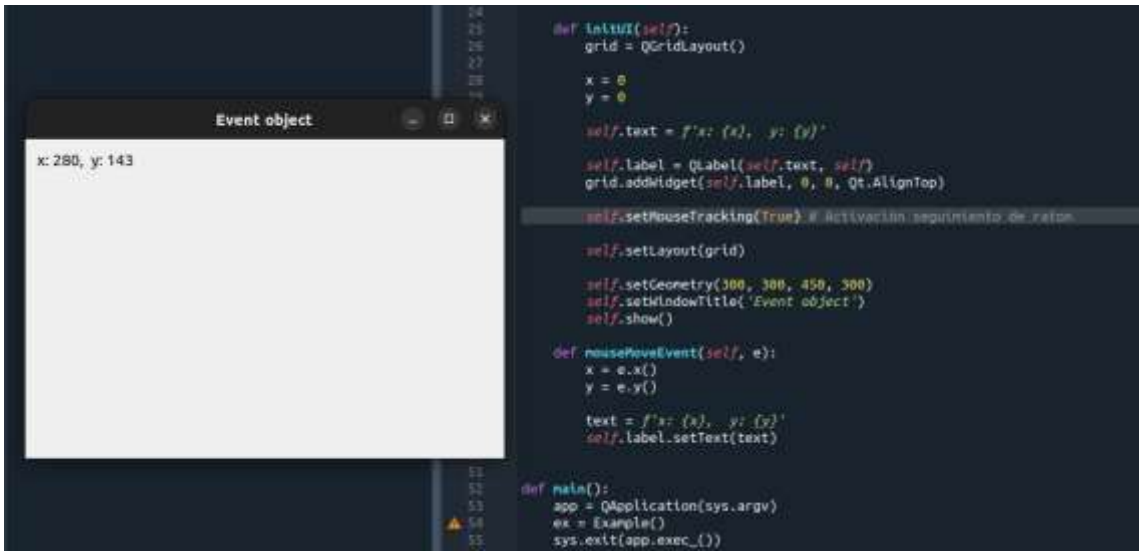

Imagen 2.1.4. Programa para ver la coordinada x e y donde se encuentra el ratón.

El código del ejemplo anterior se podrá observar en el siguiente enlace, http://www.uco.es/~i02matog/practica1/p1_ejemplo12_mapapixel.txt.

Para el siguiente ejemplo, utilizaremos dos botones, a los cuales asociaremos el método *buttonClicked* al cual irá asociado una determinada acción cada vez que hagamos click en el botón. Ambos botones están conectados a un mismo evento *clicked.connect(self.buttonClicked)*, es decir, ambos botones están conectados a la misma ranura. En la barra de estado de la aplicación, mostramos la etiqueta del botón que se está presionando, a través del siguiente código, http://www.uco.es/~i02matog/practica1/p1_ejemplo13_botones.txt.

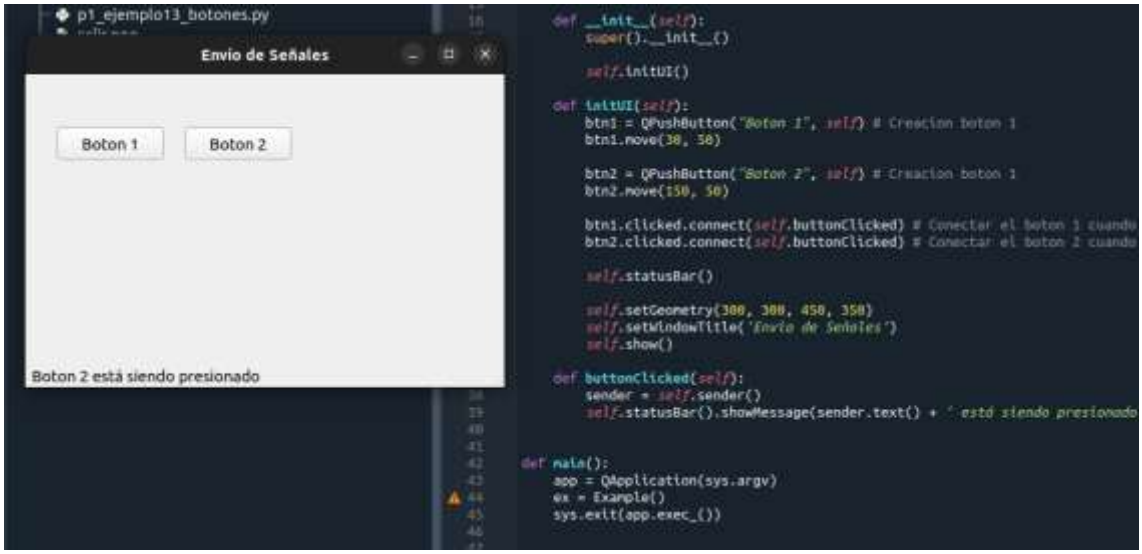

Imagen 2.1.5. Programa que activa un mensaje cuando se presiona un botón.

Los objetos creados a partir de un *QObject* pueden emitir señales. El siguiente ejemplo muestra cómo emitir señales personalizadas. Creamos una nueva señal llamada *closeApp*. Esta señal se emite durante un evento de presión del ratón. La señal se conecta a la ranura de cierre de *QMainWindow*. Se crea una señal con *pyqtSignal* como

un atributo de clase de la clase *Communicate* externa. La señal personalizada de *closeApp* está conectada a la ranura de cierre de *QMainWindow*. Cuando hacemos clic en la ventana con el puntero del ratón, se emite la señal *closeApp* y la aplicación termina.

2. Organización de Controles en las ventanas. Eventos y Slots.

Para la siguiente aplicación, se busca idear un editor de textos, que permita abrir determinados archivos y abrirlos en modo lectura, permitiendo modificar cada uno de ellos.

En primer lugar, crearemos un archivo Python a modo de archivo principal que permita crear la app e inicializarla, apareciendo en el siguiente archivo (https://www.uco.es/~i02matog/practica2/app.txt). En el siguiente archivo, tendremos un controlador que permite retornar algunos de los métodos como abrir, guardar o guardar como un determinado archivo (https://www.uco.es/~i02matog/practica2/controller.txt).

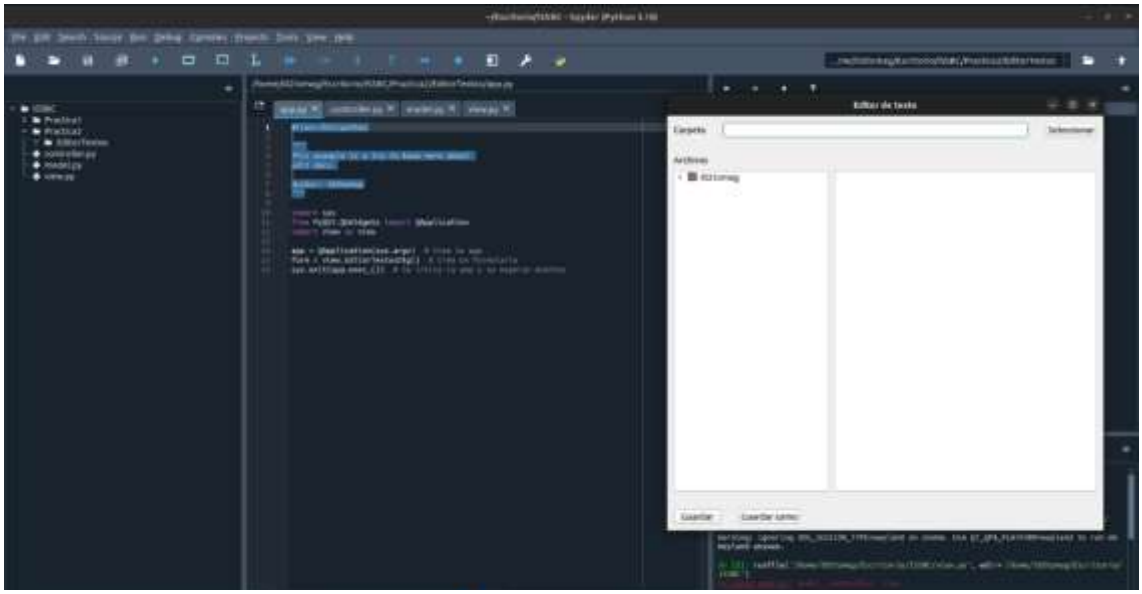

Figura 2.2.1. Archivo app.py para inicializar la app.

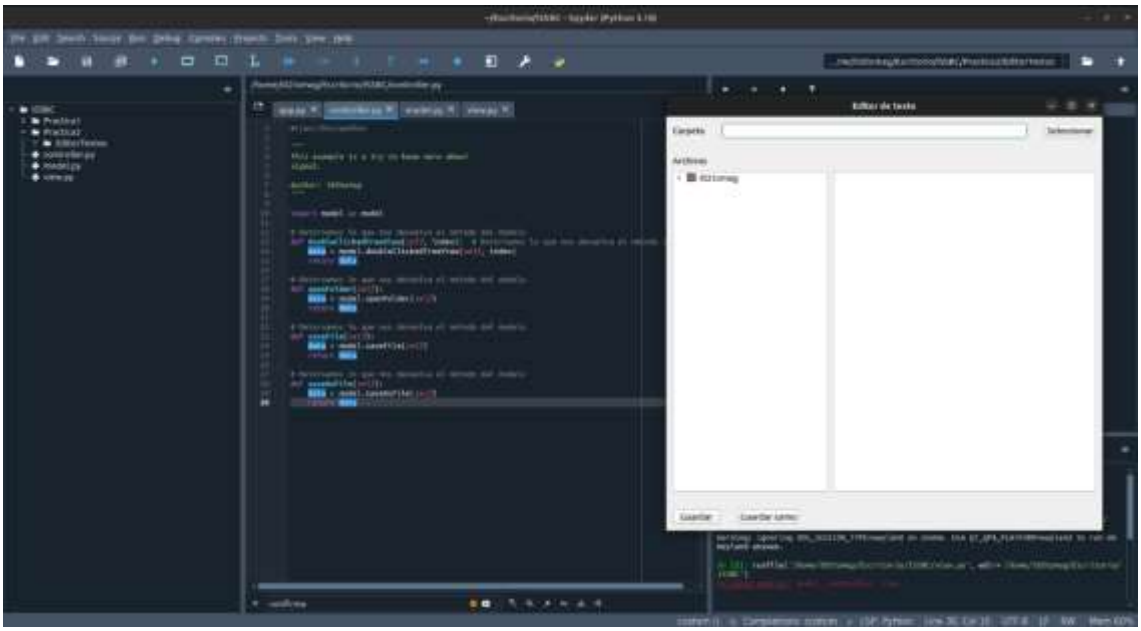

Figura 2.2.2. Archivo controller.py para retornar lo que devuelve alguno de los métodos.

En el siguiente archivo, encontraremos el archivo de vistas, donde modificaremos algunas partes de la interfaz, como la carpeta a partir de la cual comenzamos a visualizar los archivos o el título de la ventana… Además, incluiremos los botones para guardar o guardar como… Este archivo se encontrará en la siguiente ubicación, (https://www.uco.es/~i02matog/practica2/view.txt).

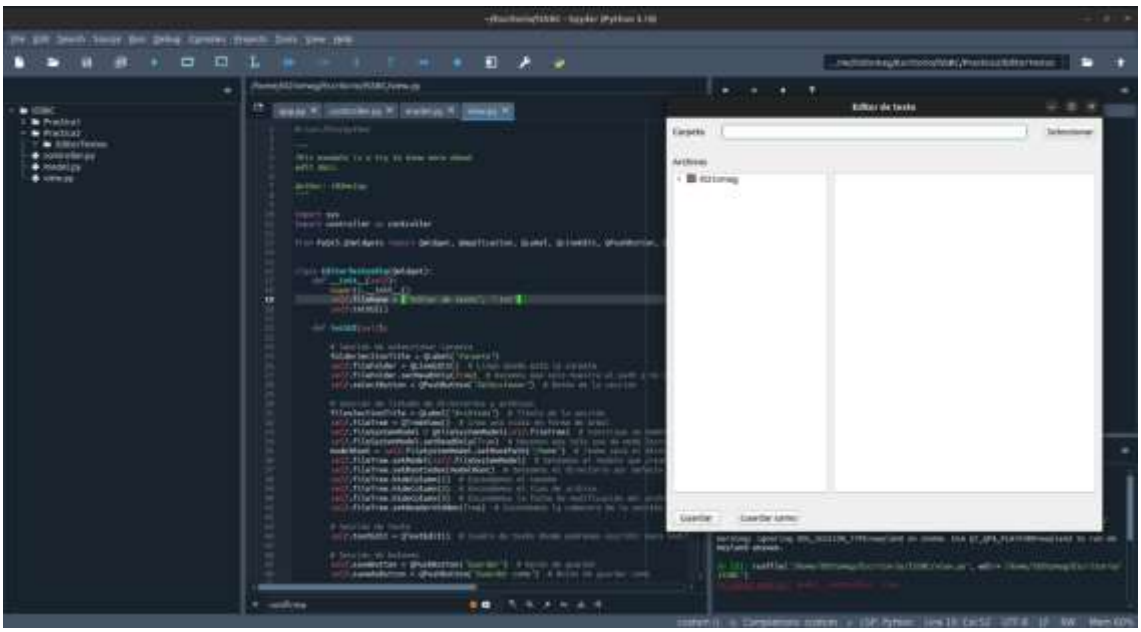

Figura 2.2.3. Archivo view.py para modificar las vistas de la app.

Por último, he creado otro archivo Python, para diseñar el modelo de la app, donde he definido cada uno de los métodos que he utilizado como guardar como, guardar o abrir archivo, como se puede observar en el siguiente link, https://www.uco.es/~i02matog/practica2/model.txt.

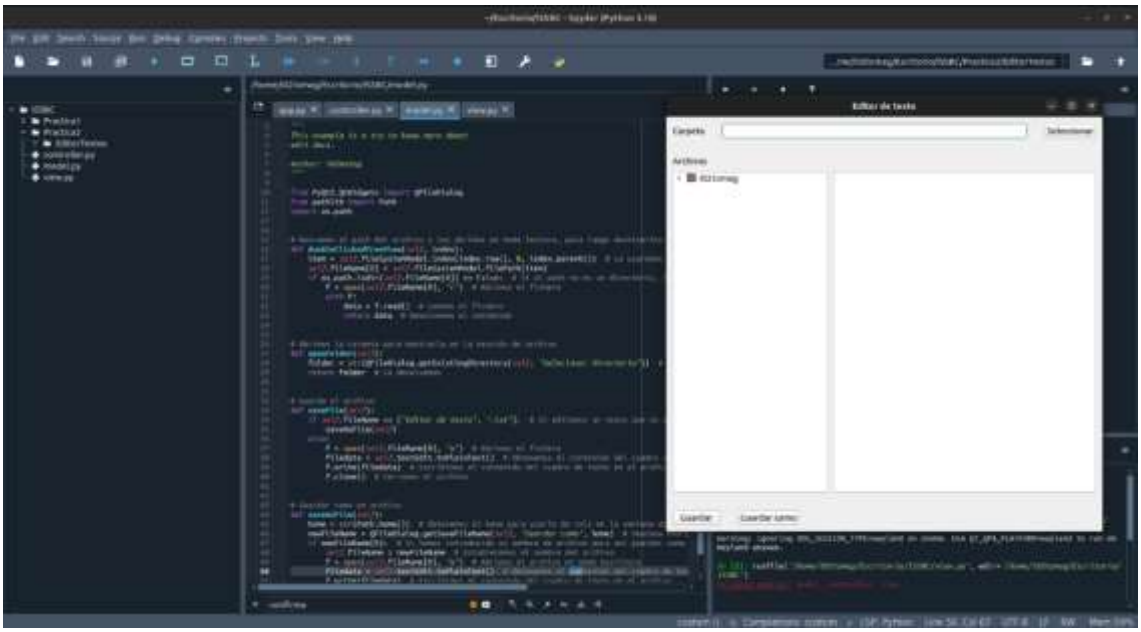

Figura 2.2.4. Archivo model.py para la definición de los métodos que utilizamos las vistas de la app.

3. Desarrollo de una Aplicación Básica.

Con los ejercicios vistos en las prácticas crear una pequeña aplicación de todos los conceptos introducidos. La aplicación debe contener lo siguiente:

- Menú de opciones.
- Barra de herramienta.
- Barra de estado.
- Una ventana con varios controles dentro.

Elaborar un informe de la práctica resumiendo los aspectos más relevantes e incluirlo en un cuaderno de prácticas. En dicho informe se establecerá hiperenlaces a su cuenta de la UCO para mostrar el código y la documentación de este.

A la hora de desarrollar la aplicación, he decidido crear una aplicación a modo de bloc de notas, que permita abrir un documento con formato .txt, partiendo de la práctica anterior. Tras utilizar este contenido, incorporaré un menú con diferentes opciones, una barra de herramientas, barra de estado y varios controles. Entre los cuales se encontrará crear un nuevo documento, abrir uno existente, guarda o salir de la aplicación.

El código de esta aplicación se encontrará dentro del siguiente enlace http://www.uco.es/~i02matog/practica2/p2_ejemplo1.txt.

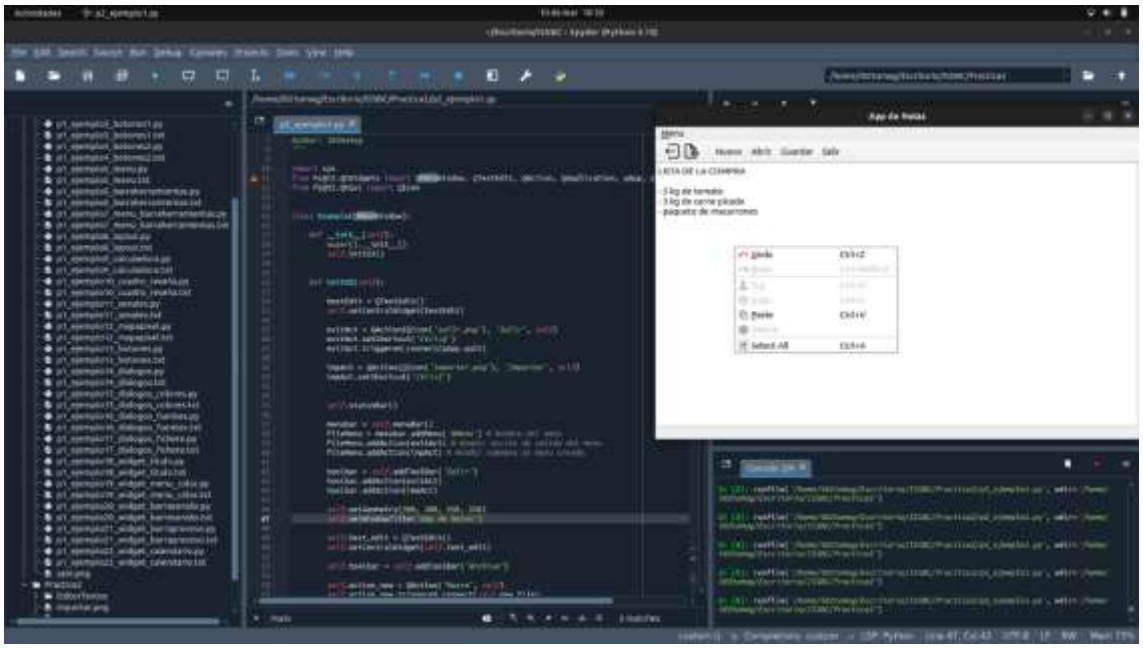

Figura 2.3.1. Archivo con la aplicación básica a modo del bloc de notas.