# Qt con Python: guía inicial


Versión 03 16/12/2020

Versión 02 10/12/2020

# Versiones

V01 - Ayoub Boujir - boujirayoub@correo.ugr.es

V02 - Francisco Llave

V03 - Prof. Andrés Roldán Aranda

V02 – 13/12/2020

# Índice

# Requerimientos

- Python 2.7 o superior y Visual Studio Code para editar código.
- Programas:
  - Qt Designer (diseño del GUI)
  - El que usamos es uno que el Prof. Andrés Roldán tiene portable preparado sobre un WinPython
- Paquetes esenciales:
  - pyqt5 (todas los servicios de Qt con Python)
- Paquetes interesantes:
  - QTdemo (para ver muchos ejemplos hechos en QT)

# ¿Cómo instalar en Python 3.7 o superior?

- Paquetes: desde la terminal (en Linux o Windows), escribir:

  pip install <nombre_paquete>

  Ejemplos:

  pip install pyqt5

  …

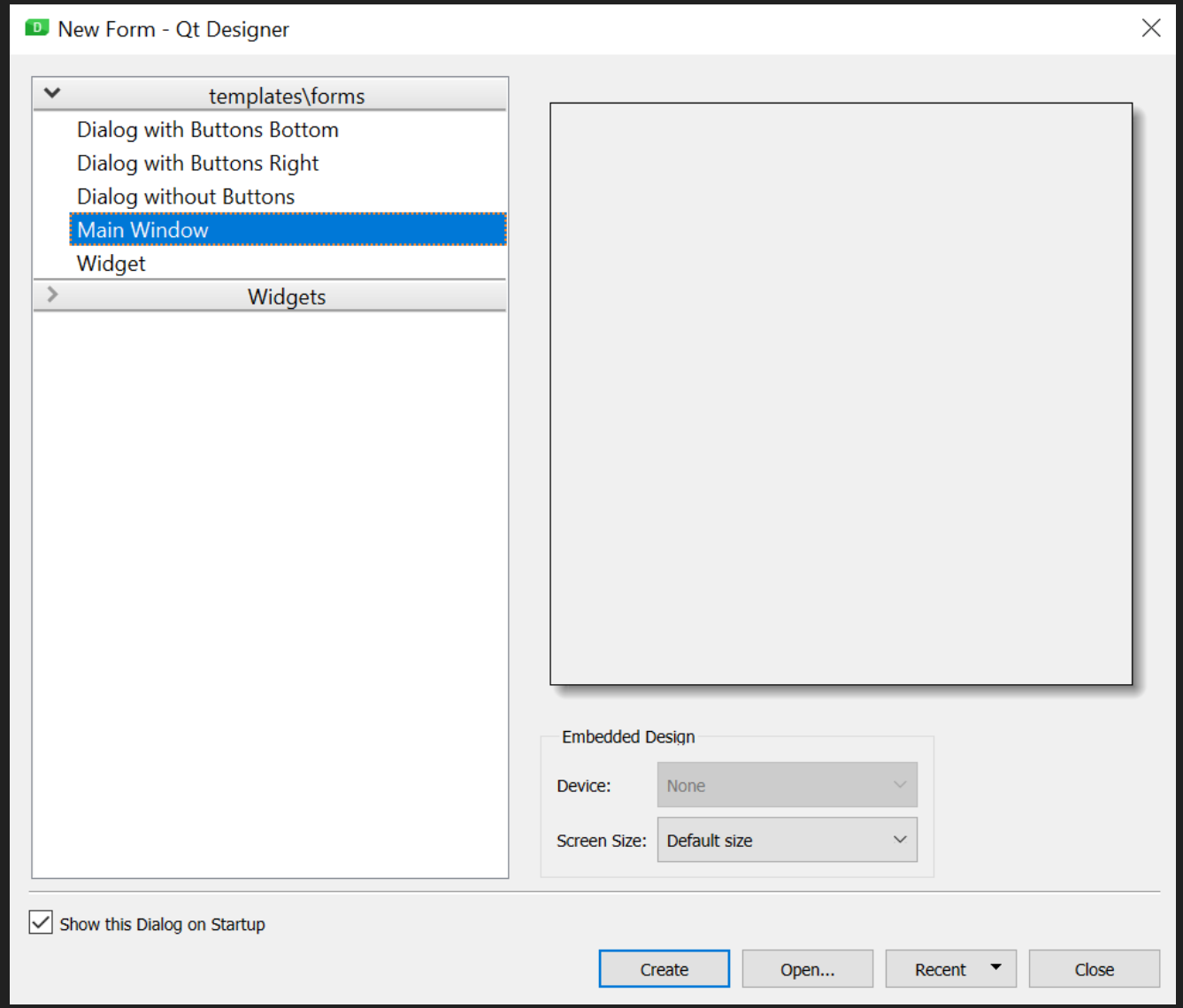
- "pip": es el gestor de paquetes de Python (se instala por defecto con Python 2.7 o posteriores)

# ¿Cómo empezar?

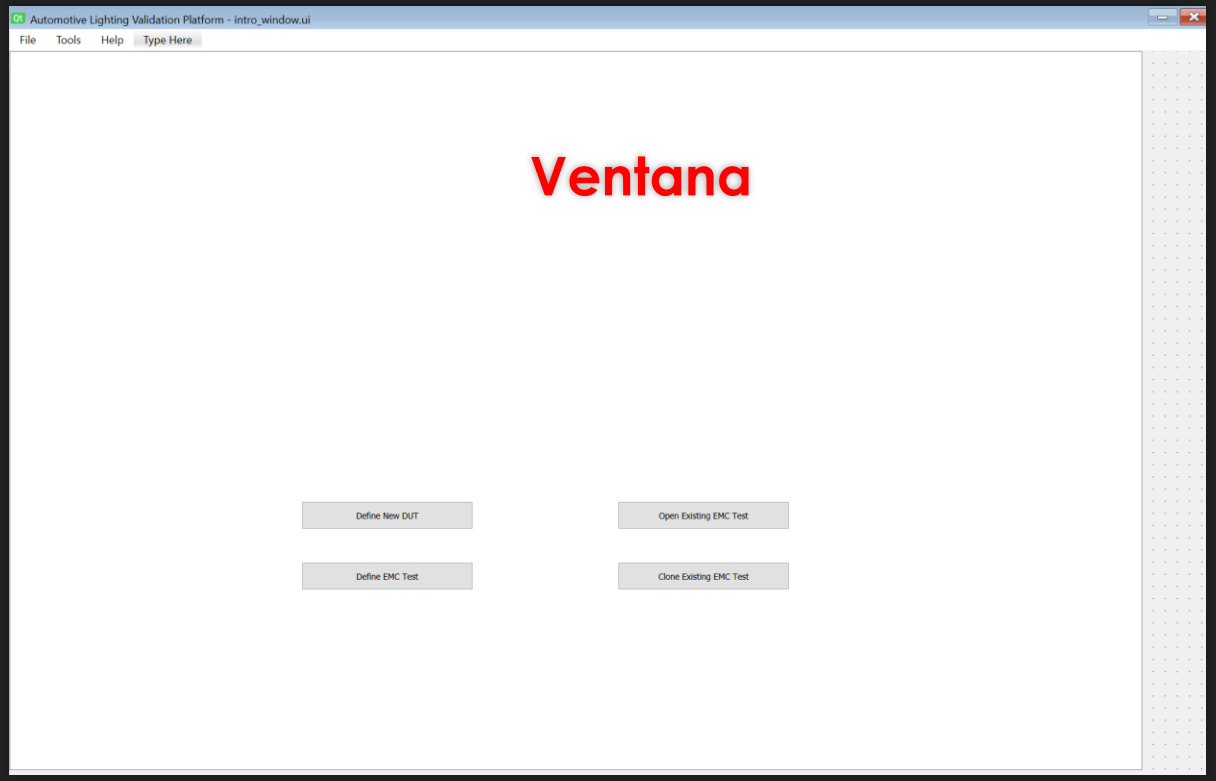
- Primero, abrir el Qt Designer desde el terminal escribiendo "designer" o usando el enlace disponible en la versión portable del WinPython.
- Realizar el diseño de tu ventana en Qt Designer y guardarlo en un archivo .ui
- Ahora, tenemos dos formas de proceder con el **interfaz gráfico**:
  - 1) Transofrmar el diseño guardado como "archivo.ui" a script de Python "archivo.py":
    - pyuic5 -x ventana.ui -o ventana_ui.py **Método no recomendable**.
  - 2) Cargar el diseño directamente con el método "loadUi" del paquete PyQt5.uic **RECOMENDADO**
- Siguiente paso sería crear nuestro propio script de Python para el **interfaz lógico**, para darle la funcionalidad que queramos a la interfaz.
- De esta manera, separamos el interfaz lógico del diseño gráfico.
- A continuación, vamos a ver un ejemplo de lo explicado anteriormente utilizando la conversión del diseño a archivo Python y otro con el método "loadUi".

# Diseño de entornos gráficos con QTDesigner

- Lo primero que debemos tener en cuenta es si queremos diseñar una Ventana o un widget.
- Una Ventana tendrá un menu, una barra de Status inferior opcional con información, etc.
- Un widget es una Ventana que no interacciona con el Sistema, si no con el programa. Realiza una acció u orden propuesta por el programa.

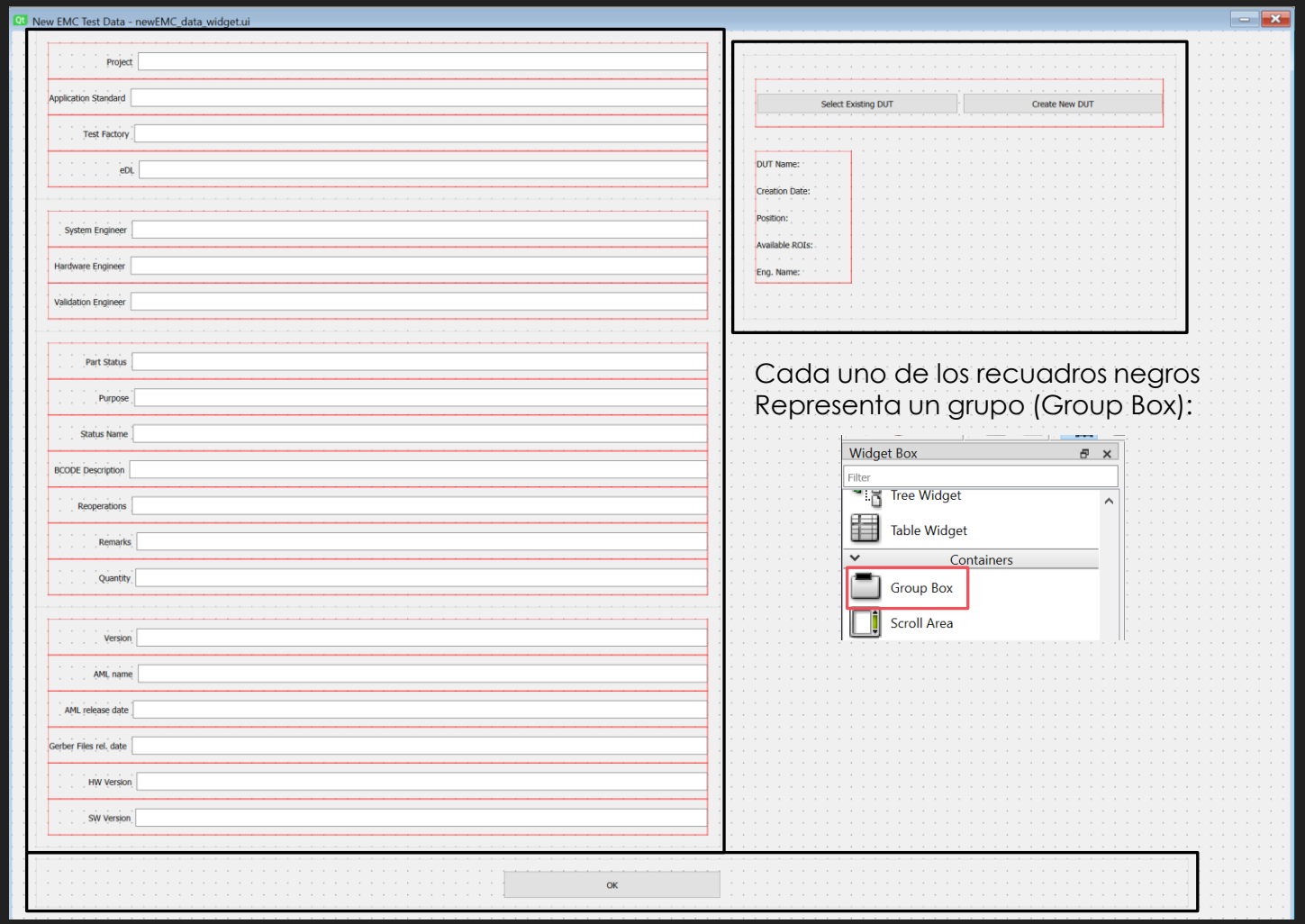


# Diferencias entre Ventana y widget:

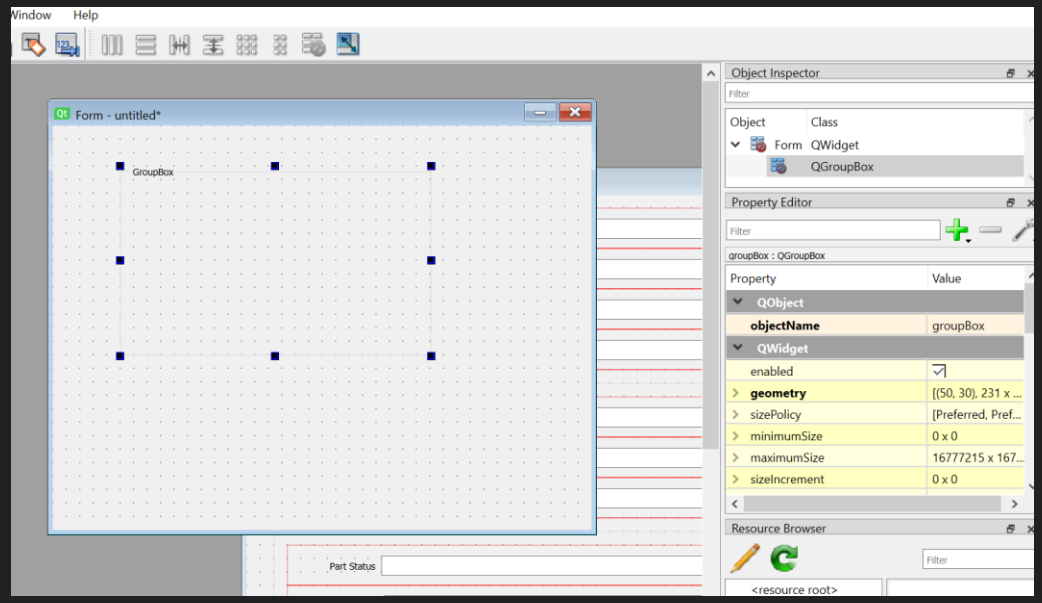



# Uso de elementos gráficos

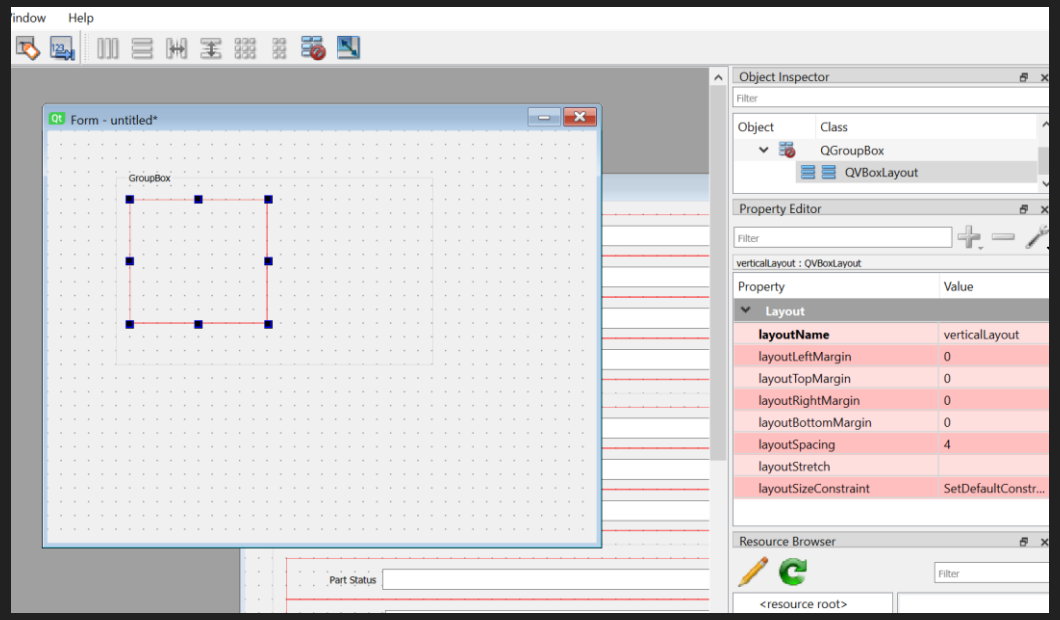
- Con el fin de insertar elementos, vamos a seguir una serie de pasos en el diseño:
  - Lo **primero** es determinar la estructura de nuestra Ventana. Si vamos a ordenar los elementos de manera **vertical** u **horizontal**.
  - Lo **segundo** sería agrupar esos elementos, si es que tendremos distintas temáticas dentro de una misma Ventana.



1



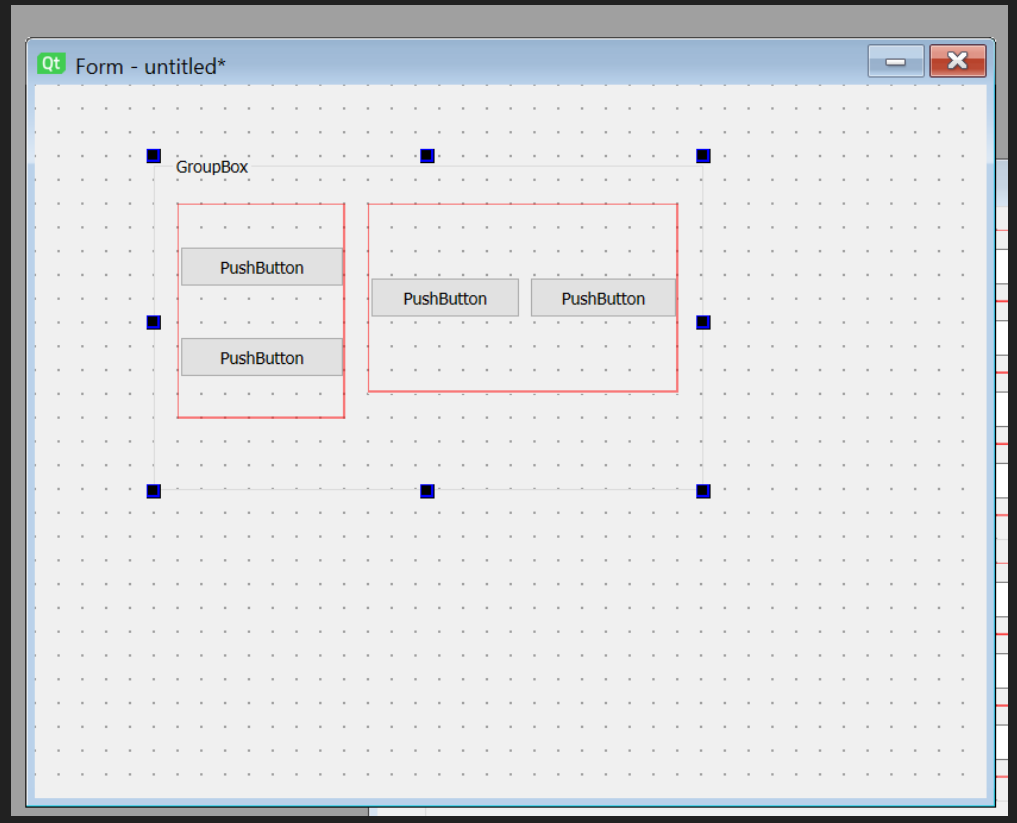

2




**Lo primero** es crear un Group Box. **Después** seleccionamos un Vertical Layout(2)
Y un Horizontal Layour a su derecha. Dentro de estos probamos a meter Push Buttons. Veremos que automáticamente se colocan de forma vertical U horizontal.

Cada Group Box se separa de otro mediante una línea. Los layout **no** van separados.

3



# ¿Cómo empezar?

1°) Diseño mi ventana en Qt Designer :


2°) Creo la clase para mi ventana principal "MainWindow":

```
class MainWindow(QtWidgets.QMainWindow):
    #MainWindow class constructor
    def __init__(self):
        #Call MainWindow super class from pyqt and load the design created in qt desginer.
        super(MainWindow,self).__init__()
        loadUi("MainWindow.ui",self)
```

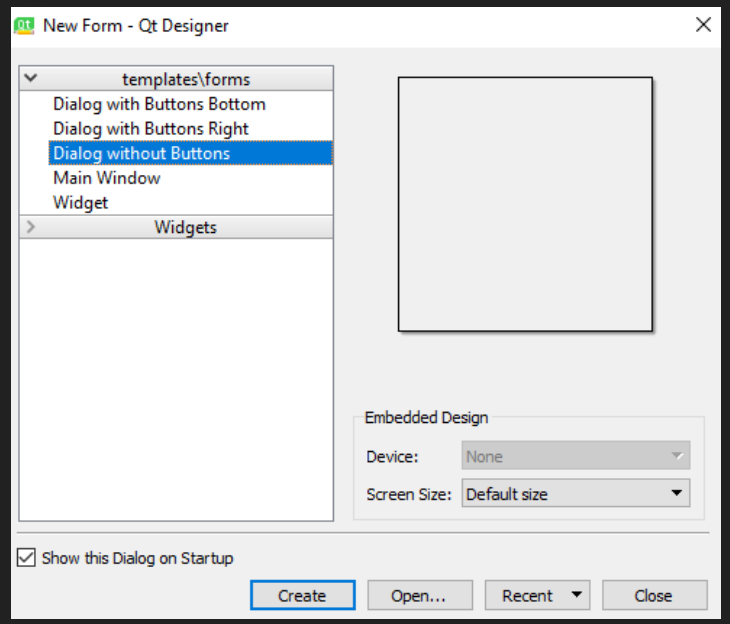
3°) Llamar y mostrar la ventana:

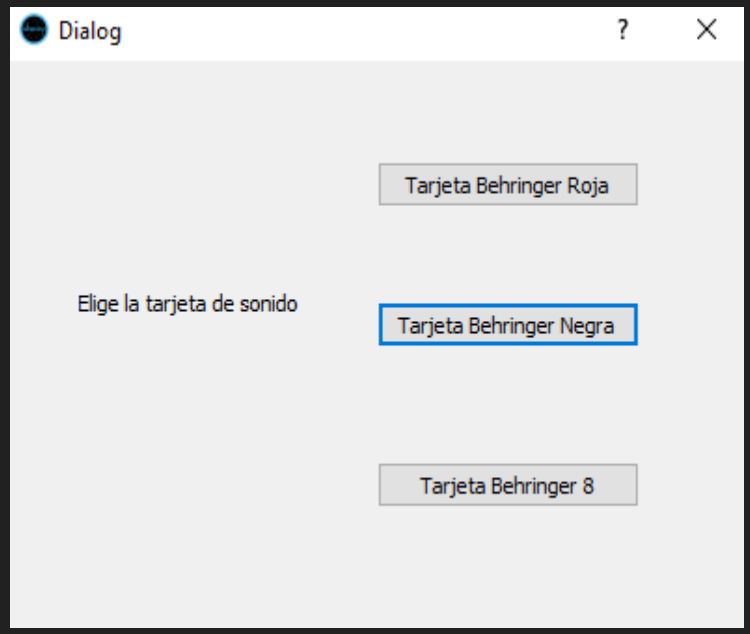
```
if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    #application = CalibrationWindow()
    application = MainWindow()
    application.show()
    sys.exit(app.exec())
```

- De esta manera no tenemos que convertir el diseño a archivo Python cada vez que hagamos una modificación en el diseño.

# ¿Cómo empezar?

- Tener en cuenta: Tipo de ventana (Llamada a la superclase):
  - En este caso, el tipo de ventana es “Qdialog”. Se llama al superconstructor “QtWidgets.QDialog”
  - En el ejemplo anterior, el tipo de ventana era “QMainWindow” y se llamó al superconstructor “QtWidgets.QMainWindow”

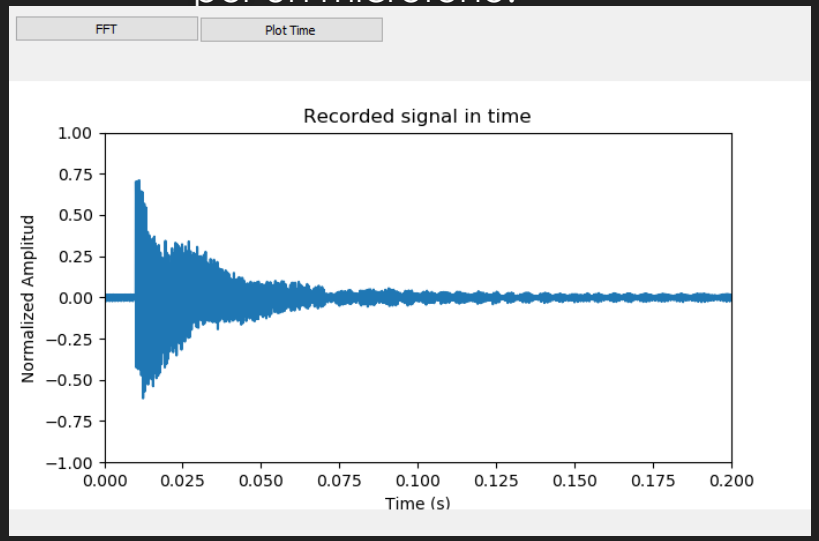


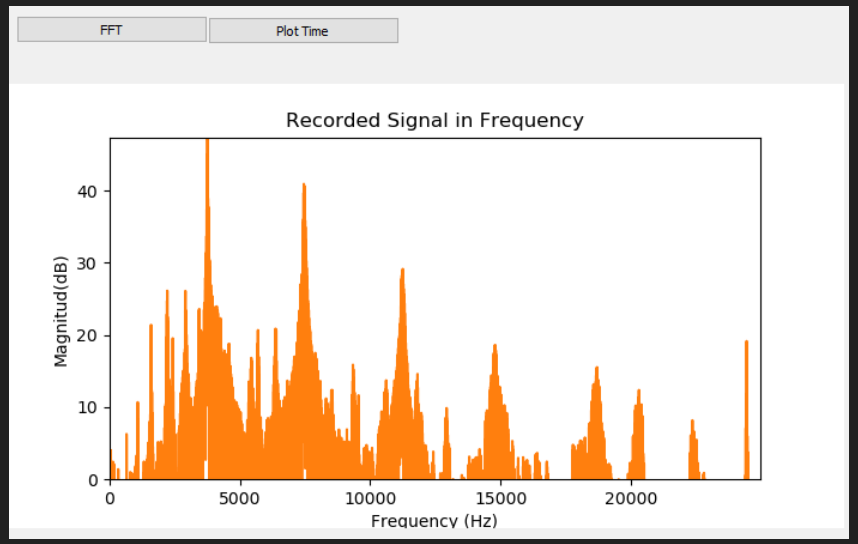




```
class ConfigWindow(QtWidgets.QDialog):
    def __init__(self,parent=None):
        super(ConfigWindow,self).__init__(parent)
        loadUi("ConfigWindow.ui",self)
```

# Funcionalidad

- 1)° Conectar el diseño con una función creada por nosotros
  - Ejemplo: Hemos creado un botón FFT que al ser pinchado me haga la FFT de la señal de audio grabada por un micrófono.




Conecto los botones FFT y Plot Time a las funciones de ploteo en tiempo y frecuencia:

```
#Connect buttons to functions
button1.clicked.connect(lambda:m.plotFrequency(wavFile))
button2.clicked.connect(lambda:m.plotTime(wavFile))
```

"boton1" y "boton2" son los nombres dados a los botones en el diseño.

# Funcionalidad

- 2)° Enlazar ventanas (para crear especie de aplicación)
  - Cada ventana diseñada en Qt Designer se carga en una nueva clase. Por ejemplo, tenemos la ventana principal "MainWindow" y la de configuración "ConfigWindow"

```
class MainWindow(QtWidgets.QMainWindow):
    #MainWindow class constructor
    def __init__(self):
        super(MainWindow,self).__init__()
        loadUi("MainWindow.ui",self)
```

```
class ConfigWindow(QtWidgets.QDialog):
    def __init__(self,parent=None):
        super(ConfigWindow,self).__init__(parent)
        loadUi("ConfigWindow.ui",self)
```

  - Creamos una función que pueda abrir la ventana ConfigWindow (ventana de configuración)

```
def openConfigWindow(self):
    new_Window=ConfigWindow(self)
    new_Window.show()
```

  - Conectamos el diseño con la función "openConfigWindow".

```
self.actionOpen_config_window.triggered.connect(self.openConfigWindow)
```

"actionOpen_config_window": Es el nombre dado a la pestaña " configuracion" de nuestro diseño en Qt Designer

# Funcionalidad

- 3)° Crear Layouts (salidas por pantalla) y la función "addWidget"
  - Con la función "addWidget" (objeto), puedo añadir un widget u objeto al Layout. De esta manera pongo el widget justo donde yo quiero.
  - Ejemplo:
    - 1° Plotear un archivo de audio en mi display (zona de mi interfaz para mostrar gráficas)

```
# Plot
m = PlotwavFile(wavFile)
if(m.maximo>1):
    QMessageBox.question(self,"Clipping","Signal is clipped, Pess OK to see signal...",QMessageBox.Ok)
#We add the plotted figure m into the corresponding layout widget in qt designer named "mplvl"
self.mplvl.addWidget(m)
```

"mplvl" : es el nombre dado en Qt Designer al layout de plotear graficas.

    - 2° Mostrar **botones**

```
# We add some buttons for FFT and to go back in windows.
button1 = QPushButton('FFT', self)
button2 = QPushButton('Plot Time',self)
self.FFT_button.addWidget(button1)
self.PlotTime_button.addWidget(button2)
```

"FFT_button" y "PlotTime_button" son los layout para los botones 1 y 2
Para ploteo en FFT y en tiempo.

# Funcionalidad

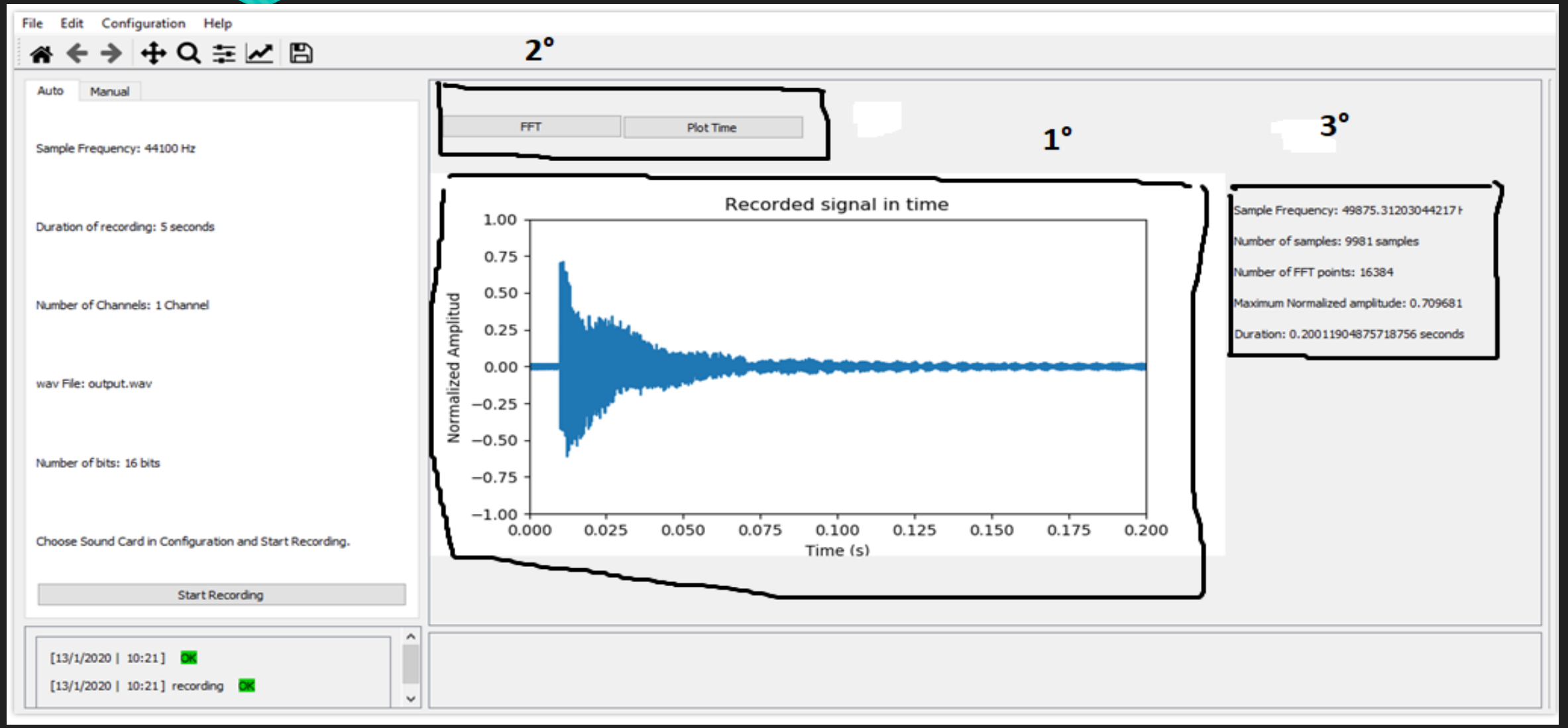
- 3° Mostrar **especificaciones** de la grafica al lado

```
# Add some important data on diplay window
label_freqsamp=QLabel()
label_freqsamp.setText("Sample Frequency: "+str(m.freq_samp)+" Hz")
self.notes.addWidget(label_freqsamp)

label_length=QLabel()
label_length.setText("Number of samples: "+str(m.Length_audioSignal)+" Samples")
self.notes_2.addWidget(label_length)

label_numberofpoints=QLabel()
label_numberofpoints.setText("Number of FFT points: "+str(m.nfft2_points)+" points")
self.notes_3.addWidget(label_numberofpoints)
```

- "notes", "notes_2" "notes_3" …

Son los nombres dados a los layouts en el diseño para cada especificación.

# Funcionalidad



# Funcionalidad

- 4°) Leer y plotear archivos “.wav” y “.mat”
  - Con el método “getOpenFileName” del paquete “QtWidgets.QFileDialog” de pyqt5 podemos abrir archivos .mat y .wav desde la interfaz.

```
def getFile(self):
    filePath, _ = QtWidgets.QFileDialog.getOpenFileName(self, 'Open file','',"*.wav, *.mat")
```

  - Depende del archivo abierto, se plotea de una manera u otra.

```
if ".wav" in filePath:
    wavFile=str(filePath)
    m=PlotwavFile()
    m.plotFrequency_wavFile(wavFile)
    m.plotTime_wavFile(wavFile)

    self.mplvl.addWidget(m)
```

```
if ".mat" in filePath:
    # Is executed when we open a .mat data file
    matFile = scipy.io.loadmat(str(filePath), squeeze_me=True)
    data=matFile['A']
    Ts=matFile['Tinterval']
    signal_length=matFile['Length']
    duration=Ts*signal_length
    freq_samp=1/Ts
    m=PlotmatFile()
    m.plotFrequency_matFile(data,freq_samp,signal_length)
    m.plotTime_matFile(data,freq_samp,signal_length)

    self.mplvl.addWidget(m)
```

  - Finalmente, conectamos el diseño con las funciones de abrir ficheros .wav y .mat
    - En la clase “MainWindow” conecto la pestaña “open” y “new” de mi diseño a la función “getFile”

```
self.actionNew.triggered.connect(self.getFile)
self.actionOpen.triggered.connect(self.getFile)
```