

DESARROLLO DE INTERFACES



Adrián Solís León 2ºDAM

11/11/2025

QRadioButton:	1
QTabWidget:	3
QProgressBar:	6
QDateTimeEdit:	11
QSlider:	13
QDial:	15
Webgrafía:	17

QRadioButton:

Se usa principalmente para seleccionar una única opción entre varias.

Algunos de sus métodos son:

.setChecked(bool): Para que al arrancar la aplicación el botón esté activado o no.

.isChecked(): Devuelve un booleano, dependiendo si el botón está activado o no.

.text(): El texto del botón de radio.

Señal más frecuente:

toggled(bool): Se emite cuando el botón cambia de estado, true si está marcado o false si lo desmarcan.

Adrián Solís León 2ºDAM

The screenshot shows the PyCharm IDE interface with the following details:

- File Explorer:** Shows a tree view of files under "PYTHON" and "Solis_Leon_Adrian_T2.1".
- Code Editor:** Displays the file `01_qradiobutton.py` containing Python code for a QRadioButton application.
- Run Output:** A window titled "Función ACTIVA" shows a QRadioButton labeled "Activar función" which is checked.
- Terminal:** Shows the command `PS C:\Users\alumno\Desktop\Python> & C:\Python\Python313\python.exe c:/Users/Solis_Leon_Adrian_T2.1/01_qradiobutton.py`.
- Status Bar:** Shows the line number (Lín. 18), column (col. 29), and encoding (UTF-8).

```
#Adrián Solís León 2ºDAM
from PySide6.QtCore import Qt
from PySide6.QtWidgets import QApplication, QRadioButton, QMainWindow

class VentanaPrincipal(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("Mi Aplicacion")

        widget = QRadioButton("Activar función")
        widget.isChecked()

        widget.toggled.connect(self.show_state)
        self.setCentralWidget(widget)

    def show_state(self, a):
        if (a == True):
            self.setWindowTitle("Función ACTIVA")
        else:
            self.setWindowTitle("Función DESACTIVADA")

app = QApplication([])
window = VentanaPrincipal()
window.show()
app.exec()
```

The screenshot shows the PyCharm IDE interface. The left sidebar displays a file tree with several Python files under 'PYTHON' and 'Solis_Leon_Adrian_T2.1'. The main editor window contains the code for '01_qradiobutton.py'. A floating window titled 'Función DESACTIVADA' shows a radio button labeled 'Activar función'. The bottom status bar shows the file path 'C:\Users\alumno\Desktop\Python & C:\Python\Python313\python.exe c:/Users/rian_T2.1/01_qradiobutton.py', line 18, column 29, and the date/time '06/11/2025 9:49'.

```
# Adrián Solís León 2ºDAM
from PySide6.QtCore import Qt
from PySide6.QtWidgets import QApplication, QRadioButton, QMainWindow

class VentanaPrincipal(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("Mi Aplicacion")
        widget = QRadioButton("Activar función")
        widget.isChecked()

        widget.toggled.connect(self.show_state)
        self.setCentralWidget(widget)

    def show_state(self, a):
        if (a == True):
            self.setWindowTitle("Función ACTIVA")
        else:
            self.setWindowTitle("Función DESACTIVADA")

app = QApplication([])
window = VentanaPrincipal()
window.show()
app.exec()
```

QTabWidget:

Se usa principalmente para agrupar el contenido en pestañas.

Algunos de sus métodos son:

.addTab(widget, “Título”): Agrega una pestaña con la página y las etiquetas dadas al widget de la pestaña y devuelve el índice de la pestaña en la barra de pestañas.

.setCurrentIndex(int): Convierte el widget en el widget actual. El widget utilizado debe ser una página en este widget de pestaña.

.count(): Devuelve los widgets que se muestran en la pestaña o ninguno.

Señal más frecuente:

currentChanged(int): Se emite cada vez que cambia el índice de la página actual, devuelve la nueva posición del índice o -1 si no hay nueva.

The screenshot shows the PyCharm IDE interface. On the left is the file browser with a tree view of Python files. In the center is a code editor with the file `02_qtabwidget.py` open, containing the following code:

```
#Adrián Solís León 2ºDAM
from PySide6.QtWidgets import QApplication, QMainWindow, QTabWidget
class VentanaPrincipal(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Página 1")
        self.tabs = QTabWidget()
        tab1 = QWidget()
        tab1_layout = QVBoxLayout()
        tab1_layout.addWidget.QLabel("Bienvenido"))
        tab1.setLayout(tab1_layout)
        tab2 = QWidget()
        tab2_layout = QVBoxLayout()
        tab2_layout.addWidget.QLabel("Segunda página"))
        tab2.setLayout(tab2_layout)
        tab3 = QWidget()
        tab3_layout = QVBoxLayout()
        tab3_layout.addWidget.QLabel("Tercera página"))
        tab3.setLayout(tab3_layout)
        self.tabs.addTab(tab1, "Inicio")
        self.tabs.addTab(tab2, "Página 2")
        self.tabs.addTab(tab3, "Página 3")
        self.tabs.currentChanged.connect(self.cambiar_pagina)
        self.setCentralWidget(self.tabs)
```

Below the code editor is a terminal window showing the output of the application's execution:

```
Página actual: 0
Página actual: 2
Página actual: 1
Página actual: 0
Página actual: 2
Página actual: 1
```

At the bottom, the status bar displays the file path as `C:/Users/alumno/Desktop/Python & C:/Users/alumno/AppData/Local/Programs/Python/Python313/python.exe c:/Users/alumno/Desktop/Python/Solis_Leon_Adrian_T2.1/02_qtabwidget.py`.

Adrián Solís León 2ºDAM

The screenshot shows a Python code editor interface with the following details:

- File Explorer (Left):** Shows a project structure under "PYTHON". The "Solis_Leon_Adrian_T2.1" folder contains several Python files: 01_qradiobutton.py, 02_qtabwidget.py, 03_qprogressbar.py, 04_qdatetimeedit.py, 05_qslider.py, 06_qdial.py, and apuntes_widgets.pdf. It also includes Hola.py and objective2_fase1.py through objective4_fase123.py.
- Code Editor (Center):** Displays the content of 02_qtabwidget.py. The code creates a QTabWidget with three tabs labeled "Inicio", "Página 2", and "Página 3". The "Página 2" tab contains the text "Segunda página".
- Terminal (Bottom):** Shows the output of running the script. It displays the following messages:

```
Página actual: 2
Página actual: 1
Página actual: 0
Página actual: 2
Página actual: 1
Página actual: 1
```
- Status Bar (Bottom):** Provides information about the terminal session, including the path (C:\Users\alumno\Desktop\Python), command (python.exe), file (02_qtabwidget.py), and current line (Líne. 1, col. 2).

The screenshot shows the PyCharm IDE interface. On the left, the Project Explorer displays a Python project structure with files like 01_qradiobutton.py, 02_qtabwidget.py, 03_qprogressbar.py, etc. In the center, the code editor shows Python code for creating a QTabWidget application. At the bottom, the terminal window shows the execution of the script and its output:

```
PS C:\Users\alumno\Desktop\Python> & C:/Users/alumno/AppData/Local/Programs/Python/Python313/python.exe c:/Users/alumno/Desktop/Python/Solis_Leon_Adrian_T2.1/02_qtabwidget.py
Página actual: 1
Página actual: 0
Página actual: 2
Página actual: 1
Página actual: 0
Página actual: 1
Página actual: 2
```

QProgressBar:

Se usa para mostrar el progreso de una tarea o valor.

Algunos de sus métodos son:

.setValue(int): Establece el valor de la propiedad

.setRange(min,max): Establece los valores mínimo y máximo de la barra de progreso en mínimo y máximo respectivamente.

Si el máximo es menor que el mínimo, el mínimo se convierte en el único valor legal.

Si el valor actual queda fuera del nuevo rango, la barra de progreso se restablece con reset() .

QProgressBar se puede establecer en estado indeterminado usando setRange(0, 0).

.reset(): Restablece la barra de progreso

No tiene señales propias pero sí reacciona a la de otros Widgets.

Adrián Solís León 2ºDAM

The screenshot shows a Python development environment with the following components:

- Explorador (File Explorer):** Shows a project structure under "PYTHON". The file "03_qprogressbar.py" is selected.
- Editor:** Displays the code for "03_qprogressbar.py". The code uses PySide6 to create a window with a progress bar and a timer to update it.
- Terminal:** Shows command-line output from running the script. It prints a menu and waits for user input.
- Output:** Shows the application window titled "Progreso: 0%" with a progress bar at 0%.
- Bottom Bar:** Includes icons for file operations, a search bar, and system status information (Lín. 2, col. 1, Espacios: 4, UTF-8, CRLF, Python 3.13.0, 8:47, 11/11/2025).

```
#Adrián Solís León 2ºDAM
from PySide6.QtCore import QTimer
from PySide6.QtWidgets import QApplication, QMainWindow, QProgressBar

class VentanaPrincipal(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Progreso: 0%")
        self.progreso_actual = 0

        self.barrera = QProgressBar()
        self.barrera.setRange(0, 100)
        self.barrera.setValue(self.progreso_actual)
        self.setCentralWidget(self.barrera)

        self.timer = QTimer(self)
        self.timer.timeout.connect(self.preguntar_usuario)
        self.timer.start(2000)

    def preguntar_usuario(self):
        print("\n==== Control de progreso ===")
        print("1 : Aumentar progreso")
        print("2 : Retroceder progreso")
        print("0 : Salir")

        opcion = input("Selecciona una opción: ")

        if opcion == "1":
            self.cambiar_progreso("aumentar")
        elif opcion == "2":
            self.cambiar_progreso("disminuir")
        elif opcion == "0":
            print("Finalizando del programa...")

    def cambiar_progreso(self,accion):
        if accion == "aumentar":
            self.progreso_actual += 10
            self.barrera.setValue(self.progreso_actual)
        elif accion == "disminuir":
            self.progreso_actual -= 10
            self.barrera.setValue(self.progreso_actual)
        else:
            self.barrera.setValue(0)
            self.setWindowTitle("Progreso: 0%")
```

```
PS C:\Users\alumno\Desktop\Python> & C:/Users/alumno/AppData/Local/Programs/Python/Python313/python.exe c:/Users/alumno/Desktop/Python/Solis_Leon_Adrian_T2.1/03_qprogressbar.py
Selecciona una opción: & C:/Users/alumno/AppData/Local/Programs/Python/Python313/python.exe c:/Users/alumno/Desktop/Python/Solis_Leon_Adrian_T2.1/03_qprogressbar.py
PS C:\Users\alumno\Desktop\Python> & C:/Users/alumno/AppData/Local/Programs/Python/Python313/python.exe c:/Users/alumno/Desktop/Python/Solis_Leon_Adrian_T2.1/03_qprogressbar.py
==== Control de progreso ===
1 : Aumentar progreso
2 : Retroceder progreso
0 : Salir
Selecciona una opción: 
```

Adrián Solís León 2ºDAM

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- File Explorer (Left):** Shows a project structure under "EXPLORADOR" with a "PYTHON" folder containing files like "01_qradiobutton.py", "02_qtabwidget.py", "03_qprogressbar.py" (selected), "04_qdatetimeedit.py", "05_qslider.py", "06_qdial.py", "apuntes_widgets.pdf", "Hola.py", "objetivo2_fase1.py", "objetivo2_fase2.py", "objetivo2_fase3.py", "objetivo2_fase4.py", "objetivo2_fase5.py", "objetivo3_fase123.py", "objetivo4_fase123.py", "objetivo5_fase12.py", "objetivo6_fase12.py", "objetivo7_fase123_ejercicio1.py", "objetivo7_fase123_ejercicio2.py", "Prueba.py", and "Solis_Leon_Adrian_T2.1.zip".
- Code Editor (Center):** Displays the content of "03_qprogressbar.py". The code creates a QMainWindow with a QProgressBar and a QTimer to handle user input for increasing or decreasing the progress.
- Terminal (Bottom):** Shows the output of running the script: "PS C:\Users\alumno\Desktop\Python> & C:/Users/alumno/AppData/Local/Programs/Python/Python313/python.exe c:/Users/alumno/Desktop/Python/Solis_Leon_Adrian_T2.1/03_qprogressbar.py". It prints the control options and the user's selection.
- Output (Bottom):** Shows the progress bar window titled "Progreso: 40%" with a progress bar at 40% completion.
- Bottom Bar:** Includes icons for file operations, a search bar, and the Python extension status.

Adrián Solís León 2ºDAM

The screenshot shows the Visual Studio Code (VS Code) interface with the following details:

- File Explorer (Left):** Shows a project structure under "EXPLORADOR" with a "PYTHON" folder containing files like "01_qradiobutton.py", "02_qtabwidget.py", "03_qprogressbar.py" (selected), "04_qdatetimeedit.py", "05_qslider.py", "06_qdial.py", "apuntes_widgets.pdf", "Hola.py", "objetivo2_fase1.py", "objetivo2_fase2.py", "objetivo2_fase3.py", "objetivo2_fase4.py", "objetivo2_fase5.py", "objetivo3_fase123.py", "objetivo4_fase123.py", "objetivo5_fase12.py", "objetivo6_fase12.py", "objetivo7_fase123_ejercicio1.py", "objetivo7_fase123_ejercicio2.py", "Prueba.py", and "Solis_Leon_Adrian_T2.1.zip".
- Code Editor (Center):** Displays the content of "03_qprogressbar.py". The code creates a QMainWindow with a QProgressBar and a QTimer to update its value.
- Terminal (Bottom):** Shows the output of running the script: "PS C:\Users\alumno\Desktop\Python> & C:/Users/alumno/AppData/Local/Programs/Python/Python313/python.exe c:/Users/alumno/Desktop/Python/Solis_Leon_Adrian_T2.1/03_qprogressbar.py", followed by a menu of options (1: Aumentar progreso, 2: Retroceder progreso, 0: Salir), the user's selection ("2"), and the current progress ("Progreso actual: 20%").
- Output (Bottom):** Shows the command line interface with "powershell" and "Python" tabs selected.
- Bottom Status Bar:** Includes icons for file operations, a search bar, and the date/time (11/11/2025, 8:47).

The screenshot shows a Python development environment with the following details:

- File Explorer:** Shows a project structure under "PYTHON". The current file is "03_qprogressbar.py". Other files include "01_qradiobutton.py", "02_qtabwidget.py", "04_qdatetimeedit.py", "05_qslider.py", "06_qdial.py", "apuntes_widgets.pdf", "Hola.py", "objetivo2_fase1.py", "objetivo2_fase2.py", "objetivo2_fase3.py", "objetivo2_fase4.py", "objetivo2_fase5.py", "objetivo3_fase123.py", "objetivo4_fase123.py", "objetivo5_fase12.py", "objetivo6_fase12.py", "objetivo7_fase123_ejercicio1.py", "objetivo7_fase123_ejercicio2.py", "Prueba.py", and "Solis_Leon_Adrian_T2.1.zip".
- Code Editor:** Displays the code for "03_qprogressbar.py". The code creates a main window with a progress bar and a timer to control its value.
- Terminal:** Shows the output of running the script. It prints a menu with options 1, 2, and 0, and then selects option 1 to increase the progress to 100%.
- Progress Bar Preview:** A small window titled "Tarea completada!" shows a progress bar at 100% completion.
- Bottom Status Bar:** Shows the path "PS C:\Users\alumno\Desktop\Python> & C:/Users/alumno/AppData/Local/Programs/Python/Python313/python.exe c:/Users/alumno/Desktop/Python/Solis_Leon_Adrian_T2.1/03_qprogressbar.py", the terminal mode ("powershell"), the Python version (3.13.0), the date and time (11/11/2025 8:48), and other system information.

QDateTimeEdit:

Se usa para seleccionar fecha y hora.

Algunos de sus métodos son:

.setDateTime(QDateTime): Establece la propiedad `dateTime()`.

.dateTime(): Obtiene la propiedad `dateTime`

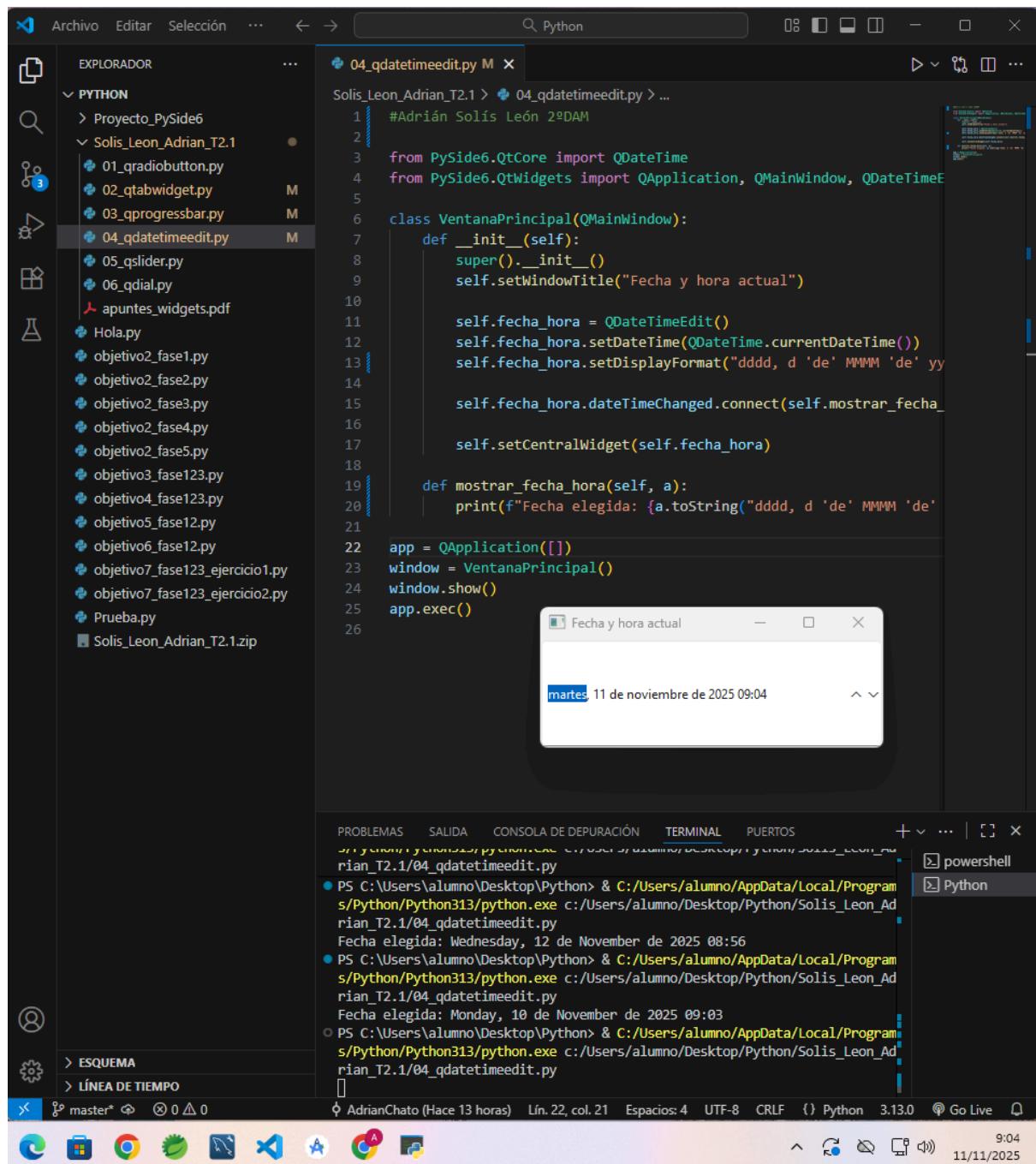
.setDisplayFormat(str): Configura la propiedad displayFormat

Señales más frecuentes:

dateChanged(QDate): Se emite cada vez que se cambia de fecha, la nueva fecha se pasa por la entrada.

timeChanged(QTime): Se emite cada vez que se cambia de hora, la nueva hora se pasa por la entrada.

dateTimeChanged(QDateTime): Se emite cada vez que cambie fecha o hora, se pasa por la entrada los nuevos datos.



```
#Adrián Solís León 2ºDAM
from PySide6.QtCore import QDateTime
from PySide6.QtWidgets import QApplication, QMainWindow, QDateTimeEdit

class VentanaPrincipal(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Fecha y hora actual")

        self.fecha_hora = QDateTimeEdit()
        self.fecha_hora.setDateTime(QDateTime.currentDateTime())
        self.fecha_hora.setDisplayFormat("ddd, d de MMMM 'de' yy")

        self.fecha_hora.dateTimeChanged.connect(self.mostrar_fecha_hora)

        self.setCentralWidget(self.fecha_hora)

    def mostrar_fecha_hora(self, a):
        print(f"Fecha elegida: {a.toString("ddd, d 'de' MMMM 'de' yy")}")

app = QApplication([])
window = VentanaPrincipal()
window.show()
app.exec()
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS + ...

- PS C:\Users\alumno\Desktop\Python> & C:/Users/alumno/AppData/Local/Programs/Python/Python313/python.exe c:/Users/alumno/Desktop/Python/Solis_Leon_Adrian_T2.1/04_qdatetimeedit.py

martes 11 de noviembre de 2025 09:04

master* 0 △ 0 AdrianChat (Hace 13 horas) Lín. 22, col. 21 Espacios: 4 UTF-8 CRLF {} Python 3.13.0 Go Live 9:04 11/11/2025

The screenshot shows a Python IDE interface with the following details:

- Left Sidebar (Explorador):** Shows a file tree with a folder named "PYTHON" containing several Python files (e.g., 01_qradiobutton.py, 02_qtabwidget.py, 03_qprogressbar.py, 04_qdatetimeedit.py) and other files like Hola.py, apuntes_widgets.pdf, and Solis_Leon_Adrian_T2.1.zip.
- Code Editor:** The active file is "04_qdatetimeedit.py". The code defines a main window with a QDateTimeEdit widget displaying the current date and time.
- Output Window:** Shows command-line logs for running the application multiple times, outputting the selected date and time each time.
- Bottom Taskbar:** Includes icons for various applications and the system clock showing "9:04" and the date "11/11/2025".

QSlider:

Se usa como control deslizante para valores numéricos.

Algunos de sus métodos son:

.setMinimum(): Establece el mínimo de la propiedad.

.setMaximum(): Establece el máximo de la propiedad.

.setValue(): Establece el valor de la propiedad.

.value(): Devuelve el valor de la propiedad, un entero.

Señal más frecuente:

valueChanged(int): Se emite cuando cambia el valor del control deslizante, el valor que se pasa es el nuevo y notifica el nuevo valor.

The screenshot shows the PyCharm IDE interface. On the left, the file tree displays a project named 'Solis_Leon_Adrian_T2.1' containing several Python files (01_qradiobutton.py, 02_qtabwidget.py, 03_qprogressbar.py, 04_qdatetimeedit.py, 05_qslider.py, 06_qdial.py) and a PDF file 'apuntes_widgets.pdf'. The current file is '05_qslider.py'. The code defines a 'VentanaPrincipal' window with a horizontal slider set from 0 to 100, starting at 50. The 'valueChanged' signal connects to a 'cambiar_brillo' slot, which updates the window title and prints the new value. The application runs in a terminal window, showing the title bar 'Nivel de brillo: 50%' and a slider control. The terminal also shows command-line logs for three executions of the program, each printing 'Nivel de brillo: 62%', 'Nivel de brillo: 50%', and 'Nivel de brillo: 50%' respectively. The bottom status bar shows the file path 'C:/Users/alumno/Desktop/Python/Solis_Leon_Adrian_T2.1/05_qslider.py', line 9, column 39, and the date/time '11/11/2025 9:11'.

```
# Adrián Solís León 2ºDAM
from PySide6.QtWidgets import QApplication, QMainWindow, QSlider
from PySide6.QtCore import Qt

class VentanaPrincipal(QMainWindow):
    def __init__(self):
        super().__init__()

        slider = QSlider(Qt.Horizontal)
        slider.setRange(0, 100)
        slider.setValue(50)
        slider.valueChanged.connect(self.cambiar_brillo)

        self.setCentralWidget(slider)
        self.setWindowTitle("Nivel de brillo: 50%")
        print("Nivel de brillo: 50%")

    def cambiar_brillo(self, valor):
        self.setWindowTitle(f"Nivel de brillo: {valor}%")
        print(f"Nivel de brillo: {valor}%")

app = QApplication([])
window = VentanaPrincipal()
window.show()
app.exec()
```

The screenshot shows the PyCharm IDE interface. The left sidebar displays a file tree with several Python files and a PDF document. The main editor window shows a Python script named `05_qslider.py`. The code creates a `QMainWindow` with a horizontal `QSlider` set from 0 to 100, with a value of 50. The slider's value is connected to a `cambiar_brillo` method, which updates the window title and prints the current value. A preview window shows a window titled "Nivel de brillo: 50%" with a horizontal slider bar. The bottom terminal window shows the script running and printing values from 81% to 100% as the slider is moved. The status bar at the bottom right shows the date and time.

```
# Adrián Solís León 2ºDAM
from PySide6.QtWidgets import QApplication, QMainWindow, QSlider
from PySide6.QtCore import Qt

class VentanaPrincipal(QMainWindow):
    def __init__(self):
        super().__init__()

        slider = QSlider(Qt.Horizontal)
        slider.setRange(0, 100)
        slider.setValue(50)
        slider.valueChanged.connect(self.cambiar_brillo)

        self.setCentralWidget(slider)
        self.setWindowTitle("Nivel de brillo: 50%")
        print("Nivel de brillo: 50%")

    def cambiar_brillo(self, valor):
        self.setWindowTitle(f"Nivel de brillo: {valor}%")
        print(f"Nivel de brillo: {valor}%")

app = QApplication([])
window = VentanaPrincipal()
window.show()
app.exec()
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

```
PS C:\Users\alumno\Desktop\Python & C:/Users/alumno/AppData/Local/Programs/Python/Python313/python.exe c:/Users/alumno/Desktop/Python/Solis_Leon_Adrian_T2.1/05_qslider.py
Nivel de brillo: 81%
Nivel de brillo: 85%
Nivel de brillo: 88%
Nivel de brillo: 91%
Nivel de brillo: 94%
Nivel de brillo: 96%
Nivel de brillo: 99%
Nivel de brillo: 100%
Nivel de brillo: 82%
```

Lín. 9, col. 39 Espacios: 4 UTF-8 CRLF {} Python 3.13.0 Go Live 9:12 11/11/2025

QDial:

Es parecido a QSlider pero es un control giratorio en vez de deslizante.

Algunos de sus métodos son:

`.setNotchesVisible(True)`: Establece el booleano de la propiedad.

`.setRange(min,max)`: Establece el valor mínimo y máximo del control.

.setValue(): Establece el valor de la propiedad.

Su señal más frecuente:

valueChanged(int): Se emite cuando cambia el valor del control giratorio, el valor que se pasa es el nuevo y notifica el nuevo valor.

The screenshot shows the PyCharm IDE interface. On the left, the project explorer displays a folder named 'Solis_Leon_Adrian_T2.1' containing several Python files (01_qradiobutton.py, 02_qtabwidget.py, 03_qprogressbar.py, 04_qdatetimeedit.py, 05_qslider.py, 06_qdial.py) and other files like apuntes_widgets.pdf and Hola.py. The file '06_qdial.py' is open in the editor, showing the following code:

```
#Adrián Solís León 2ºDAM
from PySide6.QtWidgets import QApplication, QMainWindow, QDial

class VentanaPrincipal(QMainWindow):
    def __init__(self):
        super().__init__()

        dial = QDial()
        dial.setRange(0, 10)
        dial.setNotchesVisible(True)
        dial.valueChanged.connect(self.cambiar_volumen)

        self.setCentralWidget(dial)
        self.setWindowTitle("Volumen: 0 / 10")

    def cambiar_volumen(self, valor):
        self.setWindowTitle(f"Volumen: {valor} / 10")
        if valor == 10:
            print("¡Volumen máximo alcanzado!")

app = QApplication([])
window = VentanaPrincipal()
window.show()
app.exec()
```

Below the editor, a terminal window shows the application's output:

```
Nivel de brillo: 94%
Nivel de brillo: 96%
Nivel de brillo: 99%
Nivel de brillo: 100%
Nivel de brillo: 82%
Nivel de brillo: 95%
```

A preview window shows a circular QDial with a slider at approximately 95%.

Adrián Solís León 2ºDAM

The screenshot shows the PyCharm IDE interface. The left sidebar displays a file tree under 'EXPLORADOR' with several Python files and a PDF file. The main editor window shows the code for '06_qdial.py'. The code creates a QDial widget with 10 notches, sets its range from 0 to 10, and connects its valueChanged signal to a slot named 'cambiar_volumen'. The slot updates the window title based on the dial's value and prints a message if the value reaches 10. The bottom right shows a screenshot of the application window titled 'Volumen: 10 / 10' with a circular dial. The bottom panel shows the terminal output and various toolbars.

```
#Adrián Solís León 2ºDAM
from PySide6.QtWidgets import QApplication, QMainWindow, QDial
class VentanaPrincipal(QMainWindow):
    def __init__(self):
        super().__init__()

        dial = QDial()
        dial.setRange(0, 10)
        dial.setNotchesVisible(True)
        dial.valueChanged.connect(self.cambiar_volumen)

        self.setCentralWidget(dial)
        self.setWindowTitle("Volumen: 0 / 10")

    def cambiar_volumen(self, valor):
        self.setWindowTitle(f"Volumen: {valor} / 10")
        if valor == 10:
            print("¡Volumen máximo alcanzado!")

app = QApplication([])
window = VentanaPrincipal()
window.show()
app.exec()
```

Webgrafía:

<https://doc.qt.io/qtforpython/>
<https://doc.qt.io/qtforpython-6/PySide6/QtWidgets/index.html>
<https://doc.qt.io/qt-6/qtdesigner-manual.html>

Apuntes QTimer del ejercicio.