**Universidad Autónoma de Tamaulipas**

**Facultad de Ingeniería Tampico**

A black background with a black square

Description automatically generated with medium confidenceA red and grey logo

Description automatically generated

**ASIGNATURA**

**Programación de Interfaces y Puertos**

6vo. Semestre – Grupo “I”

2025 -1

**TRABAJO**

**Desarrollo de Tareas e Investigaciones**

**UNIDAD**

**2 - DESARROLLO E IMPLEMENTACIÓN DE CONTROLADORES MEDIANTE INTERFACES Y PUERTOS**

**Docente:** Dr. García Ruiz Alejandro H.

|  |  |
| --- | --- |
| **Integrante del Equipo** | **Nivel de Participación** |
| Izaguirre Cortes Emanuel | 33.33% |
| García Salas Yahir Misael | 33.33% |
| Turrubiates Mejia Gilberto | 33.33% |
| Total: | 100% |

Indice

1.- ¿Qué es PyQt, y como se integra con Python? 2

2-. Concepto de Signal en PyQt 3

3.- Concepto de Slot en PyQt 4

4.- Comando Pyrcc5 6

5.- Comando Pyuic56.- Push Button 7

7.- Label 7

8.- Line Edit 8

9.- QMessageBox 10

10.- MainWindow vs Widget vs Dialog 10

12. Dial 12

13. QSpinBox 12

14.- QDoubleSpinBox 13

15.- Radio Button en PyQt (QRadioButton) 14

16.- CheckBox en PyQt (QCheckBox) 15

17.- ComboBox en PyQt (QComboBox) 16

18.- ListWidget en PyQt (QListWidget) 17

19.- QPixmap en PyQt 18

20.- QTimer en PyQt 19

21.- QLCDNumber en PyQt 20

Fuentes consultadas 21

# 1.- ¿Qué es PyQt, y como se integra con Python?

**¿Qué es PyQt?**

PyQt es un conjunto de enlaces de Python para la biblioteca de interfaz gráfica de usuario (GUI) Qt. Es desarrollado por **Riverbank Computing** y permite crear aplicaciones de escritorio multiplataforma con Python, utilizando las herramientas y características de Qt.

Qt es un framework ampliamente utilizado en C++ para desarrollar interfaces gráficas modernas y responsivas. PyQt proporciona una forma de acceder a esta funcionalidad desde Python, lo que permite desarrollar aplicaciones con una apariencia profesional sin necesidad de escribir código en C++.

**¿Cómo se integra con Python?**

PyQt se integra con Python a través de módulos y clases que permiten manejar eventos, interfaces gráficas y lógica de negocio. Se usa junto con **Qt Designer**, una herramienta que facilita la creación de interfaces mediante un editor visual.

**Pasos para usar PyQt en Python**

1. **Instalar PyQt**  
   Se puede instalar usando pip:

Texto

El contenido generado por IA puede ser incorrecto.

2.- **Crear una ventana básica con PyQt** Un ejemplo simple de una ventana en PyQt6:

Texto

El contenido generado por IA puede ser incorrecto.

# 2-. Concepto de Signal en PyQt

**¿Qué es un Signal en PyQt?**

En **PyQt**, un **Signal (señal)** es un mecanismo de comunicación que permite que los objetos envíen eventos o notificaciones a otros objetos sin necesidad de que estos estén directamente acoplados. Es parte del **sistema de señales y slots** de Qt, que se usa para manejar eventos en las aplicaciones gráficas.

Los **Signals** se emiten cuando ocurre un evento específico, como hacer clic en un botón o cambiar el texto de un cuadro de entrada.

**Uso de Signals en PyQt**

Los Signals están predefinidos en los widgets de PyQt, pero también podemos crear nuestros propios Signals.

**Ejemplo 1: Uso de Signals predefinidos**

Los widgets de PyQt6 tienen Signals incorporados, como clicked en un botón:

Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

En este caso, cuando se presiona el botón, se imprime el mensaje en la consola.

**Creación de Signals personalizados**

Podemos definir nuestros propios Signals utilizando pyqtSignal dentro de una clase que herede de QObject.

**Ejemplo 2: Signal personalizado**

Texto

El contenido generado por IA puede ser incorrecto.

En este caso, cuando se ejecuta emitir\_signal(), el Signal envía un mensaje que es capturado e impreso en la consola.

# 3.- Concepto de Slot en PyQt

**¿Qué es un Slot en PyQt?**

En **PyQt**, un **Slot** es una función o método que recibe y maneja una señal (**Signal**). Los Slots se usan para definir qué acción debe ejecutarse cuando se activa un Signal.

El **sistema de Signals y Slots** es la forma en que Qt maneja eventos de manera eficiente y flexible. Los Signals emiten eventos, y los Slots responden a esos eventos.

**Uso de Slots en PyQt**

Los Slots pueden ser:

1. **Métodos predefinidos** de widgets.
2. **Funciones personalizadas** creadas por el usuario.
3. **Slots personalizados** declarados explícitamente con @pyqtSlot.

**Ejemplo 1: Uso de Slots con Signals predefinidos**

Los widgets de PyQt tienen Slots predefinidos.  
Ejemplo con un botón (QPushButton) y su Signal clicked:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

**Ejemplo 2: Definir un Slot personalizado con @pyqtSlot**

Podemos usar el decorador @pyqtSlot para definir un Slot más optimizado:

Texto

El contenido generado por IA puede ser incorrecto.

# 4.- Comando Pyrcc5

**¿Qué es el comando pyrcc5?**

El comando **pyrcc5** es una herramienta de la biblioteca **PyQt5** (o PyQt6 en versiones posteriores) que se utiliza para convertir archivos de recursos de Qt (generalmente archivos .qrc) en módulos Python que pueden ser importados y utilizados en tu código.

Qt permite usar recursos como imágenes, iconos, archivos de sonido, y otros archivos estáticos dentro de las aplicaciones. Estos archivos pueden ser almacenados en un archivo .qrc (Qt Resource Collection). El comando **pyrcc5** convierte estos archivos .qrc en código Python que se puede importar para acceder a estos recursos en tu aplicación.

Sintaxis Básica:



**Explicación de los parámetros:**

* archivo.qrc: El archivo de recursos de Qt que contiene la lista de archivos que deseas incluir en tu aplicación.
* -o archivo\_rc.py: El archivo Python de salida, donde se generará el código que permite acceder a los recursos.

**¿Por qué usar pyrcc5?**

* **Encapsulación de recursos:** Permite empaquetar archivos dentro de tu aplicación sin tener que preocuparte de gestionar la ruta de los recursos manualmente.
* **Distribución más limpia:** Al incluir los recursos directamente en el código, puedes distribuir tu aplicación sin necesidad de archivos adicionales.

# 5.- Comando Pyuic56.- Push Button

El comando **pyuic5** (en el caso de PyQt5) se utiliza para convertir archivos de diseño de interfaces gráficas creados con **Qt Designer** (archivos .ui) en código Python.

Si tienes un archivo .ui que contiene un **Push Button** (botón), puedes usar pyuic5 para generar un archivo .py que represente esa interfaz gráfica.

**Comando pyuic5**

Para convertir un archivo .ui que contiene un **Push Button** en código Python, el comando sería el siguiente:



# 7.- Label

En PyQt, un **QLabel** es un widget utilizado para mostrar texto o imágenes dentro de una ventana. Es uno de los widgets más simples y se usa comúnmente para etiquetas, descripciones o para mostrar contenido estático en la interfaz gráfica.

**¿Cómo usar QLabel en PyQt?**

A continuación, te muestro cómo usar un **QLabel** en una aplicación básica de PyQt. Vamos a crear una interfaz con un **QLabel** que muestra texto y una imagen.

Ejemplo 1: QLabel con texto

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

**Propiedades comunes de QLabel:**

1. **setText(texto)**: Establece el texto que se mostrará en el QLabel.
2. **setPixmap(pixmap)**: Establece la imagen que se mostrará en el QLabel.
3. **setAlignment(align)**: Establece la alineación del texto o la imagen dentro del QLabel.
   * Ejemplo: label.setAlignment(Qt.AlignCenter) para centrar el contenido.
4. **setWordWrap(True/False)**: Habilita o deshabilita el ajuste de palabras (útil cuando se usa texto largo).
   * Ejemplo: label.setWordWrap(True).

# 8.- Line Edit

En **PyQt**, un **QLineEdit** es un widget utilizado para permitir que el usuario ingite texto de una sola línea. Es comúnmente usado en formularios, cuadros de búsqueda y en cualquier lugar donde se necesite capturar texto de una sola línea.

**¿Cómo usar QLineEdit en PyQt?**

A continuación, te muestro algunos ejemplos de cómo usar un **QLineEdit** en una aplicación PyQt.

**Ejemplo 1: QLineEdit básico**

Este es un ejemplo simple de un **QLineEdit** donde el usuario puede ingresar texto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**Explicación:**

* **QLineEdit(ventana)**: Crea un campo de entrada de texto en la ventana.
* **layout.addWidget(line\_edit)**: Agrega el campo de entrada al layout.
* El usuario puede escribir en el campo de texto, pero aún no hay funcionalidad asociada.

**Ejemplo 2: Obtener el texto de QLineEdit**

Ahora, agregamos un botón que obtiene el texto que el usuario ingresa en el **QLineEdit** y lo imprime cuando se hace clic.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

# 9.- QMessageBox

**Objetivo**: QMessageBox se utiliza para mostrar cuadros de diálogo emergentes (pop-ups) que informan al usuario sobre algo, le hacen una pregunta o le piden confirmación. Es ideal para mostrar mensajes de advertencia, error, información o preguntas.

**Signals**:

* accepted: Se emite cuando el usuario hace clic en "Aceptar".
* rejected: Se emite cuando el usuario hace clic en "Cancelar" o cierra el cuadro de diálogo.

**Diseño**: Este componente tiene botones predefinidos, como "OK", "Cancelar", "Sí", "No", dependiendo del tipo de mensaje que se quiera mostrar.

Imagen de la pantalla de un celular

El contenido generado por IA puede ser incorrecto.

# 10.- MainWindow vs Widget vs Dialog

* **QMainWindow**:
  + Es una ventana principal que proporciona una estructura básica para aplicaciones de escritorio con menús, barras de herramientas, barras de estado, etc. QMainWindow es ideal para aplicaciones con una interfaz compleja.
* **QWidget**:
  + Es la clase base para todos los componentes gráficos (widgets) en PyQt. Se puede usar para crear ventanas, botones, etiquetas, entre otros. QWidget no tiene las características adicionales que tiene QMainWindow, como la barra de menús o herramientas.
* **QDialog**:
  + Es un cuadro de diálogo, similar a QMessageBox, pero con más flexibilidad. Los diálogos pueden ser modales o no modales y se pueden personalizar más fácilmente, agregando widgets y botones según sea necesario.

Tabla

El contenido generado por IA puede ser incorrecto.

**11. Horizontal/Vertical Slider**

**Objetivo**: Los sliders (QSlider) permiten a los usuarios seleccionar un valor dentro de un rango moviendo un control deslizante, ya sea horizontal o vertical.

**Signals**:

* valueChanged(int): Se emite cada vez que cambia el valor del slider.

**Diseño**:

* QSlider puede ser horizontal o vertical dependiendo de la orientación que se le dé.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

# 12. Dial

**Objetivo**: QDial es un control circular que permite al usuario seleccionar un valor moviendo una aguja (similar a un velocímetro). Es útil cuando el rango de valores no es muy grande.

**Signals**:

* valueChanged(int): Se emite cada vez que el valor del dial cambia.

**Diseño**:

* Imagen de la pantalla de un celular

  El contenido generado por IA puede ser incorrecto.El dial puede ser configurado con un rango de valores, y los usuarios giran el control en forma de círculo para elegir un valor.

# 13. QSpinBox

**Objetivo**: QSpinBox es un widget que permite al usuario ingresar un valor numérico en un rango determinado mediante un campo de texto y botones para incrementar o disminuir el valor.

**Signals**:

* valueChanged(int): Se emite cada vez que cambia el valor del spin box.

**Diseño**:

* Es un widget de entrada numérica con botones de flecha para ajustar el valor hacia arriba o hacia abajo.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

# 14.- QDoubleSpinBox

**Objetivo**: QDoubleSpinBox es similar a QSpinBox, pero permite ingresar valores decimales en lugar de enteros. Es útil cuando se necesita ingresar números con precisión decimal.

**Signals**:

* valueChanged(double): Se emite cada vez que cambia el valor del double spin box.

**Diseño**:

* Similar a QSpinBox, pero con soporte para valores decimales.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

# 15.- Radio Button en PyQt (QRadioButton)

El **QRadioButton** es un widget de PyQt que permite seleccionar una opción entre varias dentro de un grupo. Solo un botón puede estar activo a la vez dentro del mismo grupo.

**1. Objetivo**

El objetivo del **QRadioButton** es proporcionar una opción de selección exclusiva dentro de un conjunto de opciones, lo que es útil para formularios, configuraciones y menús donde el usuario debe elegir solo una opción.

**2. Signals (Señales)**

Algunos de los signals más comunes en **QRadioButton** incluyen:

* toggled(bool): Se activa cuando el estado del botón cambia (seleccionado o deseleccionado).
* clicked(): Se emite cuando el botón es presionado.

**3. Diseño**

El **QRadioButton** se puede diseñar utilizando QGroupBox o QButtonGroup para agruparlos y asegurarse de que solo uno pueda estar activo al mismo tiempo. También se pueden personalizar con estilos mediante QSS (Qt Style Sheets).

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

# 16.- CheckBox en PyQt (QCheckBox)

El **QCheckBox** es un widget de PyQt que permite seleccionar múltiples opciones de forma independiente. A diferencia del **QRadioButton**, los checkboxes no son mutuamente excluyentes, lo que significa que el usuario puede marcar más de uno a la vez.

**1. Objetivo**

El objetivo del **QCheckBox** es permitir que los usuarios seleccionen múltiples opciones dentro de una lista, proporcionando una forma flexible de entrada en formularios, configuraciones y menús.

**2. Signals (Señales)**

Algunas señales comunes en **QCheckBox** incluyen:

* toggled(bool): Se emite cuando el estado del checkbox cambia (marcado o desmarcado).
* stateChanged(int): Se activa cuando cambia el estado del checkbox, devolviendo un valor:
  + 0: Desmarcado (**Qt.Unchecked**)
  + 1: Parcialmente marcado (**Qt.PartiallyChecked**, solo en tri-state)
  + 2: Marcado (**Qt.Checked**)

**3. Diseño**

* Se pueden organizar en un **QVBoxLayout** o **QHBoxLayout** para mantener una disposición limpia.
* Se pueden personalizar con **QSS (Qt Style Sheets)** para cambiar colores, tamaño y estilos.
* Se pueden configurar en **modo tri-state** (tres estados: marcado, desmarcado e intermedio).

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**4. Casos de Uso**

* Formularios donde el usuario puede seleccionar múltiples opciones (intereses, preferencias, configuraciones).
* Listas de selección en configuraciones de software.
* Opciones de filtrado en aplicaciones.

# 17.- ComboBox en PyQt (QComboBox)

El **QComboBox** es un widget de PyQt que permite seleccionar una opción de una lista desplegable. Es útil cuando se quiere ahorrar espacio en la interfaz, proporcionando una forma compacta de mostrar múltiples opciones.

**1. Objetivo**

El objetivo del **QComboBox** es proporcionar un menú desplegable en el que el usuario pueda elegir una opción de una lista predefinida. Se usa en formularios, configuraciones y menús donde se requiere una única selección.

**2. Signals (Señales)**

Algunas señales comunes en **QComboBox** incluyen:

* currentIndexChanged(int): Se activa cuando cambia la opción seleccionada, devolviendo el índice.
* currentIndexChanged(str): Se activa cuando cambia la opción seleccionada, devolviendo el texto de la opción.
* activated(int): Se emite cuando el usuario selecciona una opción (clic o teclado).
* highlighted(int): Se emite cuando el usuario resalta una opción sin seleccionarla.

**3. Diseño**

* Se puede poblar manualmente con addItem() o addItems().
* Se puede establecer una opción predeterminada con setCurrentIndex().
* Se puede hacer editable con setEditable(True), permitiendo al usuario escribir su propia opción.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**4. Casos de Uso**

* Formularios donde se requiere elegir una opción (país, categoría, tipo de usuario).
* Menús de configuración en aplicaciones.
* Filtros en listas y tablas dinámicas.

# 18.- ListWidget en PyQt (QListWidget)

El **QListWidget** es un widget de PyQt que permite mostrar una lista de elementos, donde los usuarios pueden seleccionar uno o varios elementos. A diferencia de **QComboBox**, este widget es ideal cuando se quiere mostrar todas las opciones disponibles de manera visible sin necesidad de desplegarlas.

**1. Objetivo**

El objetivo del **QListWidget** es proporcionar una interfaz para mostrar una lista de elementos de forma interactiva, permitiendo selecciones únicas o múltiples.

**2. Signals (Señales)**

Algunas señales comunes en **QListWidget** incluyen:

* itemClicked(QListWidgetItem): Se emite cuando un usuario hace clic en un elemento.
* itemDoubleClicked(QListWidgetItem): Se emite cuando un usuario hace doble clic en un elemento.
* currentItemChanged(QListWidgetItem, QListWidgetItem): Se activa cuando el usuario cambia de selección.
* itemSelectionChanged(): Se emite cuando la selección en la lista cambia.

**3. Diseño**

* Se pueden agregar elementos con addItem() o addItems().
* Se puede configurar el **modo de selección** con setSelectionMode(), permitiendo seleccionar uno o varios elementos:
  + QAbstractItemView.SingleSelection: Solo un elemento a la vez.
  + QAbstractItemView.MultiSelection: Selección múltiple permitida.
  + QAbstractItemView.ExtendedSelection: Selección múltiple con Shift/Ctrl.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**4. Casos de Uso**

* Listas de selección en aplicaciones (por ejemplo, seleccionar un archivo o categoría).
* Menús donde los usuarios deben elegir entre múltiples opciones visibles.
* Interfaces que requieren selección múltiple de elementos.

# 19.- QPixmap en PyQt

El **QPixmap** es una clase de PyQt diseñada para manejar imágenes y gráficos de manera eficiente. Se utiliza principalmente para mostrar imágenes en **QLabel** u otros widgets, y es más eficiente que **QImage** en términos de rendimiento gráfico.

**1. Objetivo**

El objetivo de **QPixmap** es cargar, manipular y mostrar imágenes en una aplicación PyQt de forma optimizada. Es ideal para trabajar con gráficos en la interfaz de usuario.

**2. Métodos Principales**

Algunos métodos comunes en **QPixmap** incluyen:

* load(path): Carga una imagen desde un archivo.
* scaled(width, height): Escala la imagen a un tamaño específico.
* save(path, format): Guarda la imagen en un archivo.
* fromImage(QImage): Convierte un **QImage** en un **QPixmap**.

**3. Ejemplo de Uso**

**Logotipo

Descripción generada automáticamente**

# 20.- QTimer en PyQt

El **QTimer** es una clase de PyQt que permite ejecutar funciones de manera repetitiva o después de un cierto tiempo. Es útil para actualizar interfaces gráficas, crear temporizadores, ejecutar tareas en segundo plano y animaciones sin bloquear la interfaz.

**1. Objetivo**

El **QTimer** se usa para programar tareas con un intervalo de tiempo específico, evitando el uso de time.sleep(), que congela la interfaz gráfica.

**2. Métodos Principales**

Algunos métodos importantes en **QTimer** incluyen:

* start(ms): Inicia el temporizador con un intervalo en milisegundos.
* stop(): Detiene el temporizador.
* setInterval(ms): Establece un nuevo intervalo de tiempo.
* timeout.connect(función): Conecta la señal timeout a una función para ejecutarla cuando el temporizador se active.
* isActive(): Verifica si el temporizador está corriendo.

**3. Ejemplo de Uso Básico**

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**4. Casos de Uso**

* Actualizar relojes y temporizadores en una interfaz.
* Crear animaciones en interfaces gráficas.
* Programar eventos automáticos.
* Evitar bloqueos en la interfaz cuando se necesita ejecutar una tarea repetitiva.

# 21.- QLCDNumber en PyQt

El **QLCDNumber** es un widget de PyQt que muestra números en un estilo de pantalla LCD (dígitos de siete segmentos). Es útil para representar relojes, contadores y datos numéricos de forma visual.

**1. Objetivo**

El objetivo de **QLCDNumber** es mostrar valores numéricos de manera clara y llamativa, similar a un display digital.

**2. Métodos Principales**

Algunas funciones importantes en **QLCDNumber** incluyen:

* display(value): Muestra un número en el LCD (puede ser int o float).
* setDigitCount(n): Establece el número de dígitos visibles.
* setMode(mode): Define el sistema numérico (Decimal, Binario, Octal o Hexadecimal).
  + QLCDNumber.Dec: Decimal (por defecto).
  + QLCDNumber.Bin: Binario.
  + QLCDNumber.Oct: Octal.
  + QLCDNumber.Hex: Hexadecimal.
* setSegmentStyle(style): Cambia el estilo de los segmentos.
  + QLCDNumber.Outline: Contorno (por defecto).
  + QLCDNumber.Filled: Segmentos rellenos.
  + QLCDNumber.Flat: Estilo plano.

**3. Ejemplo de Uso**

**Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente**

# Fuentes consultadas

 French, J. (2023, 22 de mayo). *Singletons in Unity (done right)*. Game Dev Beginner. Recuperado el 13 de marzo de 2024, de https://gamedevbeginner.com/singletons-in-unity-the-right-way/#unity\_singleton

 Technologies, U. (s.f.). *Unity - Manual: Barra de control del Scene View*. Recuperado el 13 de marzo de 2024, de <https://docs.unity3d.com/es/530/Manual/ViewModes.html>

 Technologies, U. (s.f.-a). *El Character Controller (Controlador del personaje) - Unity Manual*. Recuperado el 13 de marzo de 2024, de https://docs.unity3d.com/es/2018.4/Manual/class-CharacterController.html

 Technologies, U. (s.f.-b). *Unity - Manual: Administrador de input (Input Manager)*. Recuperado el 13 de marzo de 2024, de https://docs.unity3d.com/es/530/Manual/class-InputManager.html

 Toolify AI. (2023, 11 de diciembre). *Aprende a utilizar los RAYCASTS en Unity*. Recuperado el 13 de marzo de 2024, de https://www.toolify.ai/es/ai-news-es/aprende-a-utilizar-los-raycasts-en-unity-223667