

PORTAFOLIO DE PRÁCTICA: Practica 6 -

**QPushButton, QLabel,
QFrame, QComboBox y QLineEdit**

Programación Visual

FACILITADOR:

ALI PÉREZ GÓMEZ

SEMESTRE: CUARTO SEMESTRE

INTEGRANTES:

**Lopez Fuentevilla Suheidy
López Melchor Itzel Yared
Prieto Vargas Luis Eduardo**

Índice

Introducción.....	pag 3
Objetivo	pag 4
Marco teorico.....	pag 4
Simulación	pag 9
Recolección de información.....	pag 9
Análisis de resultado.....	pag 9
Conclusiones.....	pag10
Referencia.....	pag10

Introducción

El siguiente documento tiene como fin evidenciar el reporte de las evidencias realizadas en las prácticas de Programación Visual dando una breve explicación de la practica realizada en el aula.

Como una introducción breve de lo que se podrá encontrar en este documento de evidencias será un poco del procedimiento del código en Python y su diseño en QT designer.

Objetivo

Este ejemplo muestra cómo usar botones, labels, frames, combobox y line edit para entrada de texto. Los frames son QWidgets contenedores que permiten organizar otros QWidgets dentro de ellos. Los combobox permiten tener una lista de opciones predeterminada o agregar más elementos a la lista, para después seleccionar una de todas las opciones. Los line edit son entradas de texto y los label son etiquetas..

Marco teórico

QPushButton es un componente que permite a los usuarios interactuar con la aplicación mediante clics. Es uno de los widgets más básicos y comunes en una interfaz gráfica. Cuando el usuario presiona el botón, este emite señales que pueden ser conectadas a funciones o métodos para realizar acciones específicas.

Características Principales

Texto e Íconos:

Texto: Puedes asignar un texto al botón que se muestra en su superficie.

Ícono: Puedes agregar un ícono al botón junto con el texto o solo un ícono.

Señales y Slots:

clicked(): Emitida cuando el botón es presionado y liberado.

pressed(): Emitida cuando el botón es presionado.

released(): Emitida cuando el botón es liberado.

toggled(bool checked): Emitida cuando un botón "checkable" cambia de estado.

Propiedades de Estilo:

Hojas de Estilo: Permite personalizar la apariencia del botón utilizando hojas de estilo (QSS), similar a CSS en web.

Iconos y Fuentes: Personaliza el ícono y el tipo de fuente.

Estados del Botón:

Habilitado/Deshabilitado: Controla si el botón puede ser interactuado.

Checkable: Permite que el botón tenga un estado alterno (presionado/no presionado).

QLabel es un widget de visualización que puede mostrar texto simple o enriquecido (usando HTML), así como imágenes. Este widget es útil para mostrar etiquetas, encabezados, mensajes, imágenes o cualquier contenido visual que no necesite interacción del usuario.

Características Principales

Texto e Imágenes:

Texto: Puede mostrar texto simple o texto formateado utilizando HTML.

Imágenes: Puede mostrar imágenes, ajustarlas y escalarlas según sea necesario.

Interacción Limitada:

Aunque QLabel está diseñado principalmente para mostrar información, puede detectar algunos eventos, como el paso del ratón sobre él, y actuar como un enlace cuando se habilita.

Autoajuste de Tamaño:

Se puede configurar para ajustar automáticamente su tamaño al contenido que muestra, ya sea texto o una imagen.

Estilos:

Se puede personalizar su apariencia usando hojas de estilo (QSS).

QFrame es una clase base para widgets con una apariencia de marco, como QGroupBox, QLabel, y QToolBox. Permite crear marcos alrededor de widgets, lo que ayuda a definir secciones en una interfaz gráfica.

Características Principales

Tipos de Marcos:

NoFrame: Sin marco.

Box: Marco estilo caja.

Panel: Marco estilo panel.

StyledPanel: Panel estilizado según el estilo actual.

HLine: Línea horizontal.

VLine: Línea vertical.

WinPanel: Panel estilo Windows.

Sunken: Marco hundido.

Raised: Marco elevado.

Estilos de Líneas:

Plain: Línea simple.

Raised: Línea elevada.

Sunken: Línea hundida.

Grosos de Líneas:

lineWidth: Grosor de la línea del marco.
midLineWidth: Grosor de la línea central del marco.
Color del Fondo:

Puedes configurar el color del fondo utilizando las propiedades estándar de Qt o con hojas de estilo (QSS).

QComboBox es un widget de selección que permite al usuario elegir una opción de una lista desplegable. También puede permitir la entrada de texto si está configurado para ser editable.

Características Principales

Lista Desplegable: Muestra una lista de opciones al hacer clic en la flecha de la caja combinada.

Editable: Puede configurarse para permitir que el usuario ingrese texto.

Modelos de Datos: Puede trabajar con modelos de datos para manejar grandes conjuntos de opciones.

Íconos y Texto: Cada opción en la lista puede mostrar un ícono junto con el texto.

QLineEdit es un widget de entrada de texto que permite al usuario ingresar y editar una sola línea de texto. Se utiliza comúnmente en formularios y diálogos para capturar información del usuario.

Características Principales

Entrada de Texto:

Permite la entrada de una sola línea de texto.

Puede ser configurado para aceptar solo ciertos tipos de datos, como números o direcciones de correo electrónico.

Máscaras de Entrada:

Soporta máscaras de entrada para restringir el formato del texto ingresado.

Ajustes de Texto:

Puede alinear el texto a la izquierda, al centro o a la derecha.

Permite establecer texto de sugerencia (placeholder) que aparece cuando el campo está vacío.

Validación:

Permite la validación de la entrada de texto mediante validadores.


Funciones de Edición:

Soporta operaciones básicas de edición como cortar, copiar, pegar, deshacer y rehacer.

Simulación

Software

```
# python3.py > ...
1 import sys
2 from PyQt5 import QtWidgets, QtCore
3
4 class Ventana(QtWidgets.QMainWindow):
5     def __init__(self, parent=None):
6         QtWidgets.QMainWindow.__init__(self, parent)
7         self.setWindowTitle("practica 6.2", self)
8
9         self.pushButton.clicked.connect(self.mostrar)
10        self.pushButton_2.clicked.connect(self.agregar)
11
12        for i in range(10):
13            self.comboBox.insertItem(self.comboBox.currentIndex(), "Elemento {}".format(i))
14
15
16    def agregar(self):
17        if self.lineEdit.text() != "":
18            self.comboBox.insertItem(self.comboBox.currentIndex(), self.lineEdit.text())
19            self.lineEdit.clear()
20        else:
21            pass
22
23    def mostrar(self):
24        self.label.setText("Selección: {}".format(self.comboBox.currentIndex()))
25
26
27 app = QtWidgets.QApplication(sys.argv)
28 vVentana = Ventana()
29 vVentana.show()
30 sys.exit(app.exec_())
```



Recolección de información

La paractica realizada tenia como finalidad hacer un backend y un frontend y vincularlos usando QT designer y Python con visual studio code, ademas la función de este es mostrar o plasmar un mensaje en el label que indica la actual selección del combobox. Por otro lado, la función agregar valida que en el line edit efectivamente haya algún texto, de ser así, agrega el texto respectivo a la lista de opciones del combobox

Análisis de resultado

La dificultad de la practica no es muchas sin embargo hay que tener en cuenta el nombre de cada widget que se agregue ya que en la parte del código debe llamarse igual, por lo demás no fue muy complicado.

Conclusiones

Al final el objetivo de la practica se cumplió, otra cosa que añadir es que se debe contar con las librerías necesarias para compilar el código, de lo contrario por más que el código este bien no te reconocerá las funciones.

Referencias

Blanchette, J., & Summerfield, M. (2008). C++ GUI Programming with Qt 4 (2nd ed.). Prentice Hall.

Este libro proporciona una guía exhaustiva sobre la programación de interfaces gráficas con Qt, incluyendo el uso de widgets como QLineEdit.

Molkentin, D. (2007). The Book of Qt 4: The Art of Building Qt Applications. No Starch Press.

En este libro, se exploran las características y el uso práctico de Qt, con ejemplos detallados sobre cómo trabajar con QLineEdit y otros componentes de la biblioteca.

Ezust, A., & Ezust, P. (2011). Introduction to Design Patterns in C++ with Qt 4 (2nd ed.). Prentice Hall.

Este recurso se centra en los patrones de diseño en C++ utilizando Qt, ofreciendo casos de estudio y ejemplos que incluyen la implementación de QLineEdit.