Revisión de conceptos

1. Concepto de Interfaz

La interfaz gráfica de usuario, también conocida por sus siglas en inglés GUI (*graphical user interface*), es un entorno visual e informático, conformado por elementos gráficos que ayudan y facilitan la comunicación de un sistema con su usuario o toda persona con la que interactúe.

Se puede considerar a la interfaz gráfica como la cara de un programa, ya que es lo único que puede ver el usuario y es donde podrá realizar todas las acciones disponibles en el programa. Está conformada de un conjunto de imágenes, cuadros de texto, botones, lista de opciones, etc., que tienen por objetivo entregar o adquirir algún tipo de información.

2. GUI en Python

Existen diferentes librerías en Python con el objetivo de realizar interfaces graficas de usuario. Algunas de las librerías más conocidas, y que cuentan con una buena compatibilidad de funcionamiento con las plataformas actuales, son Tkinter, Wxpython y PyQt. En este curso nos vamos a centrar en la última.

Para iniciar en la programación de una GUI en Python con PyQt es necesario definir algunos conceptos:

Widgets: también se conoce como elemento de control. Son todos aquellos elementos que interaccionan de forma directa con el usuario, como un botón, una caja de texto, una lista de opciones, etc. Su principal función es facilitar el manejo de información sobre una aplicación.

Evento: cuando el usuario realiza una acción sobre uno de los widgets en la GUI, se dice que se genera un evento. De acuerdo con el tipo de ese evento, como un clic en un botón; escribir un dato en una caja de texto; escoger una de las opciones en una lista; etc., el programa tiene la capacidad de ejecutar algún tipo de función en específico.

Señal: cuando se presenta un evento, este a su vez genera una señal característica con la que el programa puede identificar que acción se va a tomar. Un mismo widget tiene diferentes señales, por ejemplo, un botón puede tener una señal que se activa al presionarlo y liberarlo, otra distinta cuando solo se presiona o cuando solo se libera.

Slot: es la función particular que se ejecuta ante la señal generada por un evento. Por ejemplo, cuando el usuario presiona y libera un botón llamado **Cerrar** genera una señal, esta indica al programa que se debe ejecutar una función o slot encargado de cerrar la ventana actual del sistema.

3. Manejo básico de PyQt

PyQt es un enlace (*binding*) de Python con el kit de herramientas de Qt. Este hace uso de clases para un mejor manejo de los diferentes *widgets*, señales, *slots* y ventanas de una GUI. En el momento, hay dos versiones de PyQt: PyQt4 y PyQt5, que corresponden a Qt4 y Qt5. Por la compatibilidad de PyQt5 con los sistemas actuales y el hecho de que ya viene preinstalado con Anaconda, este será el que se utilizará en el curso.

A continuación, se muestra el código básico para obtener una ventana que permita visualizar una información por medio del *widget* llamado *label* o etiqueta:

```
from PyQt5 import QtWidgets # Importamos el módulo de PyQt para el menejo basico de ventanas
import sys # Modulo que permite el desarrollo del flujo principal (main loop)

class VentanaPrincipal(QtWidgets.QWidget):
    def __init__(self):
        QtWidgets.QWidget.__init__(self) # Ejecutar el constructor de la clase padre QWidget
        self.setGeometry(400, 100, 300, 80) # Definir las dimensiones de la ventana
        # (coordenada horizontal, coordenada vertical, ancho, alto)
        self.setWindowTitle("Primera Ventana")
        # Crear un Widget de tipo etiqueta
        self.etiqueta = QtWidgets.QLabel(self) # Se especifica la ventana donde se fija
        self.etiqueta.setText("Hola Mundo!!") # Se especifica el texto de la etiqueta
        self.etiqueta.move(5,10); # se especifica la posición en coordenadas de la etiqueta en la
ventana (horizontal, vertical)
        # Crear un segundo Widget de tipo etiqueta
        self.etiqueta2 = QtWidgets.QLabel(self)
        self.etiqueta2.setText("Esta es mi primera ventana")
        self.etiqueta2.move(5,40)

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv) # Maneja el flujo principal del programa y las
características principales
    # Solo puede haber uno en el programa
    ventana = VentanaPrincipal() # Se crea la ventana
    ventana.show() # Metodo para mostrar la ventana en pantalla
    sys.exit(app.exec_()) # Se ejecuta el flujo principal y se especifica cuando se acaba el programa

>>
# Se obtiene la siguiente ventana:
```

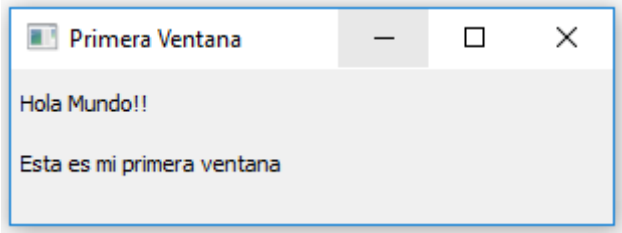

Figura 1. Primera venta con información.

Nota: para conocer más métodos de la clase Qwidget como setWindowTitle() o setGeometry(), consulte la página oficial para la documentación de PyQt5 referenciado en la sección Material de apoyo.

Ahora, se incluyen tres botones (*widgets*) a la ventana anterior y vamos a indicar a cada uno una señal de evento y su respectivo *slot* o función a ejecutar. Un botón se usará para mostrar un mensaje oculto por medio de un *slot* propio, otro botón se encargará de ocultar el mensaje con un *slot* propio y el ultimo botón cerrará la ventana por medio de un *slot* predefinido por PyQt.

```
from PyQt5 import QtWidgets
import sys

class VentanaPrincipal(QtWidgets.QWidget):
    def __init__(self):
        QtWidgets.QWidget.__init__(self)
        self.setGeometry(400, 100, 300, 130)
        self.setWindowTitle("Primera Ventana")
        self.etiqueta = QtWidgets.QLabel(self)
        self.etiqueta.setText("Hola Mundo!!")
        self.etiqueta.move(5,10)
        self.etiqueta2 = QtWidgets.QLabel(self)
        self.etiqueta2.setText("Esta es mi primera ventana")
        self.etiqueta2.move(5,40)
        # Creamos una etiqueta donde se va a mostrar el mensaje
        self.mensaje = QtWidgets.QLabel(self)
        self.mensaje.move(90,100)
        self.mensaje.resize(280,20) # Definir un tamaño  para la etiqueta para que el mensaje oculto se pueda ver
        # Crear un Widget de tipo botón, para el botón de Cerrar la ventana
        self.quit_boton = QtWidgets.QPushButton("Cerrar", self) # Se especifica el texto del botón y la ventana donde se fija
        self.quit_boton.move(5, 60) # se especifica la posición en coordenadas del botón en la ventana (horizontal, vertical)
        self.quit_boton.clicked.connect(QtWidgets.qApp.quit) # Se indica que la señal del evento es porque se presionó y soltó el botón, y la señal está asociada al slot predefinido QtWidgets.qApp.quit
```

```
        # Crear el botón de mostrar el mensaje oculto
        self.mostrar_boton = QtWidgets.QPushButton("Mostrar", self)
        self.mostrar_boton.move(100, 60)
        self.mostrar_boton.clicked.connect(self.Mostrar_Mensaje) # Se indica que la señal del evento es porque se presionó y soltó el botón, y la señal está asociada al slot o función propia Mostar_Mensaje
        # Crear el botón de ocultar el mensaje
        self.borrar_boton = QtWidgets.QPushButton("Ocultar", self)
        self.borrar_boton.move(190, 60)
        self.borrar_boton.clicked.connect(self.Ocultar_Mensaje) # Se indica que la señal del evento es porque se presionó y soltó el botón, y la señal está asociada al slot o función propia Ocultar_Mensaje

    # Función o slot a ejecutar cuando se presiona el botón de mostrar mensaje
    def Mostrar_Mensaje(self):
        self.mensaje.setText("Se presiono el botón") # Se escribe el mensaje en la etiqueta
    # Función o slot a ejecutar cuando se presiona el botón de borrar mensaje
    def Ocultar_Mensaje(self):
        self.mensaje.setText("") # Se escribe un carácter vacío en la etiqueta del mensaje

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    ventana = VentanaPrincipal()
    ventana.show()
    sys.exit(app.exec_())
```

>>

Ventana que aparece al ejecutar el código, el mensaje se encuentra oculto:

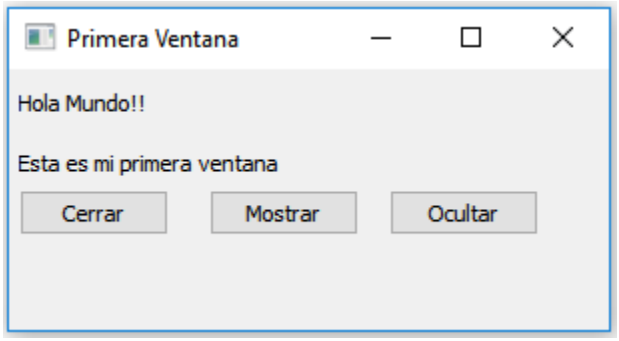

Figura 2. Ventana con los tres botones.

Ventana con el mensaje visible:

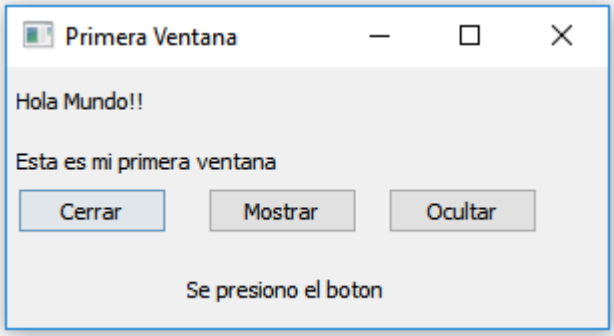

Figura 3. Ventana con los botones y una acción realizada.

Existen una variedad de *widgets* en PyQt, algunos de estos son:

***Label*:** se usa para mostrar texto al usuario.
PushButton: se usa a menudo para hacer que el programa haga algo con el usuario, simplemente presionando un botón.
RadioButtons: es un botón que puede alternar entre un estado marcado o no marcado (1 o 0) y tiene forma circular. Normalmente se usa junto con otros RadioButtons, pero en estos casos solo se puede marcar uno de ellos.
CheckBox: proporciona un estado de marcado o no marcado y tiene forma cuadrada. Comúnmente, se usa para indicar cuando una característica está habilitada.
ScrollBar: proporciona una forma para moverse horizontal o verticalmente dentro de una ventana donde el contenido es demasiado grande para ajustarse.
LineEdit: es una caja de entrada de texto (para una sola línea), utilizado para recibir la entrada de texto del usuario.

Nota: para conocer más *widgets*, además de sus respectivas señales y *slots* predefinidos, consulte la página oficial para la documentación de PyQt5 referenciada en la sección Material de apoyo.

En PyQt5 se puede utilizar la clase QDialog de QtWidgets para crear ventanas secundarias. En el próximo ejemplo se crea una ventana principal que contiene una etiqueta, una caja de texto y un botón. Al presionar dicho botón se abrirá una ventana secundaria con el mensaje ingresado en la caja de texto de la ventana principal.

```
from PyQt5 import QtWidgets
import sys

class VentanaPrincipal(QtWidgets.QWidget):
    def __init__(self):
        QtWidgets.QWidget.__init__(self)
        self.setGeometry(400, 100, 300, 130)
        self.setWindowTitle("Primera Ventana")
        self.etiqueta2 = QtWidgets.QLabel(self)
        self.etiqueta2.setText("Ingrese texto: ")
```

```
        self.etiqueta2.move(10,40)
        # Widget para la entrada de texto
        self.campo_texto = QtWidgets.QLineEdit(self)
        self.campo_texto.move(90, 37)
        # Crear el botón de abrir la ventana secundaria
        self.mostrar_boton = QtWidgets.QPushButton("Abrir Ventana Secundaria", self)
        self.mostrar_boton.move(70, 70)
        self.mostrar_boton.clicked.connect(self.Abrir_Ventana) # Indicar el slot para la señal de presionar el boton

    # Función o Slot para abrir la ventana secundaria y enviar el texto ingresado
    def Abrir_Ventana(self):
        texto_ingresado = self.campo_texto.text() # Se lee el texto ingresado
        ventana2 = VentanaSecundaria(self, txt = texto_ingresado) # Se crea el objeto o ventana secundaria, y se envía el texto como argumento
        ventana2.show(); # Se muestra en pantalla

# Clase de la ventana secundaria, se hereda de Qdialog
class VentanaSecundaria (QtWidgets.QDialog):
    def __init__(self, ppal= None, txt = ""):
        super(VentanaSecundaria,self).__init__(ppal) # Se llama el constructor de la clase padre, se envía el nombre de la clase hija
        # Se organizan los widgets y propiedades de la ventana secundaria
        self.setGeometry(400, 100, 250, 100)
        self.setWindowTitle("Ventana Secundaria")
        self.etiqueta = QtWidgets.QLabel(self)
        self.etiqueta.setText("Ventana Secundaria")
        self.etiqueta.move(65,20)
        self.etiqueta2 = QtWidgets.QLabel(self)
        self.etiqueta2.setText("El texto ingresado fue: "+ txt) # Se muestra el texto ingresado
        self.etiqueta2.move(50,40)

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    ventana = VentanaPrincipal()
    ventana.show()
    sys.exit(app.exec_())

>>
# Ventana principal que aparece al ejecutar el código:
```

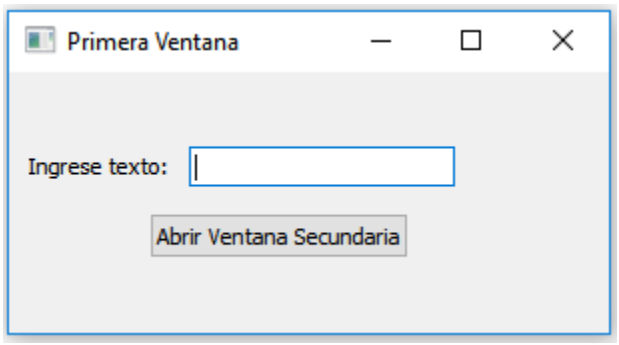

Figura 4. Ventana con caja de texto.

Ventana secundaria al presionar el botón (se ingresó a la caja de texto Hola Mundo!):

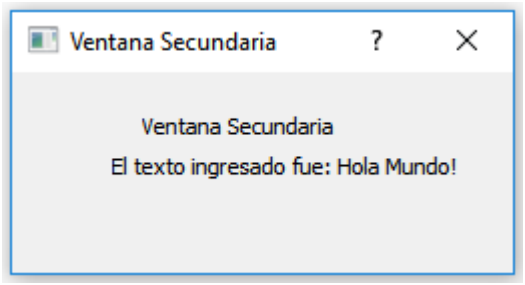

Figura 5. Ventana con el texto ingresado.

Nota: para conocer otros tipos de ventana, consulte la página oficial para la documentación de PyQt5 referenciada en la sección Material de apoyo.

Como se observa en los códigos anteriores, incluir cada uno de los *widgets* necesarios para un programa, además de organizarlos de una forma estéticamente aceptable, puede incluir muchas líneas de código. Por esto, existe una herramienta que facilita el trabajo de elaborar todo el diseño visual de las interfaces graficas con PyQt. Esta herramienta se llama Qt Designer.

3. Manejo básico de Qt Designer

Esta herramienta de Qt permite diseñar y construir interfaces gráficas de usuario para la clase QtWidgets, facilitando la composición y personalización de los distintos tipos de ventana presentes en esta clase. Todo esto permite manejar de una forma gráfica el avance en la interfaz, así el programador observa en tiempo real el estado de la interfaz.

Qt Designer posee barras de herramientas con todos los *widgets* disponibles y estos se pueden arrastrar a la ventana que se esté diseñando. La herramienta permite editar las señales y *slots* de cada uno, además existe una barra de propiedades que sirve para modificar las propiedades tanto de la ventana como de los *widgets* (como son el tamaño, nombre, color, etc). Las ventanas y *widgets* creados con Qt Designer se integran perfectamente con el código programado en Python, de modo que se pueda asignar fácilmente el comportamiento a los elementos gráficos.

4. Modelo Vista Controlador (MVC)

El MVC es un patrón de arquitectura de *software* donde las funciones del sistema se dividen en tres componentes: vistas, modelos y controladores. Igualmente, separa la lógica de funcionamiento de la lógica de la vista, permitiendo estructurar sistemas robustos de forma clara y eficiente; sobre todo pensando en que los sistemas sean escalables y requerirán mantenimiento.

Modelo: contiene los datos y la funcionalidad de la aplicación, es decir, se encarga de realizar las funciones de actualizar, búsqueda, consulta, procesamiento de datos, etc.

Controlador: determina qué procesos debe realizar el modelo cuando el usuario interacciona con el sistema. Luego, comunica a la vista los resultados.

Vista: gestiona cómo se muestran los datos en la interfaz gráfica.

En la siguiente figura se muestra un esquema de cómo interactúan cada uno de estos componentes del sistema y el usuario.

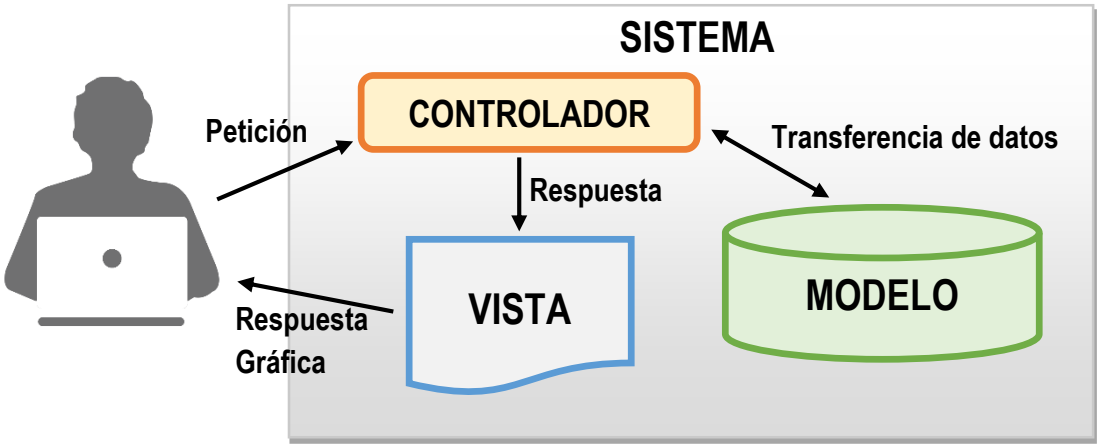

Figura 5. Interacción entre el usuario y el sistema.

El flujo de control clásico para este tipo de estructura de programación consiste en:

1. El usuario realiza una acción en la interfaz gráfica. Por ejemplo, realizar una búsqueda de datos o procesar unos datos de entrada.

2. El controlador identifica el evento de entrada. Este es generado por el usuario.

3. El controlador notifica al modelo la acción del usuario, de esa manera, el controlador realiza las acciones respectivas. Esto puede implicar un cambio del estado del modelo por la eliminación o ingreso de nuevos datos.

4. Se genera una nueva vista con la información o datos enviados por el modelo. El modelo no tiene conocimiento directo de la vista, solo procesa información de acuerdo con la acción indicada por el controlador.

5. La interfaz de usuario espera otra interacción del usuario. Con esto comenzará un nuevo ciclo.

La partición de las responsabilidades en los tres componentes permite realizar cambios en alguna parte del código sin afectar otra parte. Por ejemplo, si modificamos la Base de Datos, solo deberíamos modificar el modelo, encargado de los datos, y el resto de la aplicación debería permanecer intacta.

La separación entre la vista y el modelo permite que los diseñadores gráficos puedan dedicarse al diseño de la interfaz, a la vez que los programadores se encargan de programar y probar el modelo, sin ninguna interferencia entre ellos. Luego solo sería necesario programar el controlador adecuado y la aplicación quedaría lista.
