**Ejercicio tema 2.1**

**Caso práctico 1**

-Instalar Visual Studio Code

-Instalar Python3

-Instalar PIP (Package Installer for Python3)

-Crear un entorno visual de desarrollo para Python3

-Instalar PySide6 en el entorno visual

-Ejecutar un programa con interfaz gráfica de usuario en el entorno visual.

**PIP**

**Necesitamos preparar nuestra máquina para nuestros desarrollos, así que instalamos el intérprete Python3, que nos permitirá ejecutarlos junto con su gestor de paquetes pip (package installer for Python).**

**Solución**: Si estamos en un sistema **Linux/macOS**, en nuestra distribución ya debe estar instalado el intérprete de Python; de no ser así, podemos instalar el paquete Python3 con nuestro gestor de paquetes del sistema (apt para derivados de debian, pacman para derivados arch...).

Para sistemas **Windows**, descargamos la última versión de Python3 de la página de descargas de Python.

Es importante marcar la opción de incluir Python3 al PATH del sistema para que la consola de PowerShell reconozca posteriormente la orden desde cualquier ubicación. También dejamos marcada la opción de instalar pip.

Más tarde instalaremos el gestor de paquetes pip.

**Linux/macOS:**

$ python -m ensurepip --upgrade (python3 -m ensurepip --upgradesegun sistema operativo)

**Windows:**

C:> py -m ensurepip –upgrade

También podemos descargar el script de instalación desde

<https://bootstrap.pypa.io/get-pip.py> y ejecutarlo.

**Linux/macOS:**

$ python get-pip.py

**Windows:**

C:> py get-pip.py

Aunque no hace falta si ya lo habéis marcado durante la instalación de Python3.

Una vez está todo instalado, procedemos a la comprobación desde la consola correspondiente a nuestra plataforma.

Linux/macOS:

$python --version (o $python3 --version)

$ pip --version (o pip3 --version)

Windows:

C:> py --version

C:> pip --version

Si no reconoce pip como un comando, debemos agregar la ruta a pip (C:\Users\TuUsuario\AppData\Local\Programs\Python\Python3XX\Scripts) a la variable de entorno PATH, donde XX es nuestra versión de Python.

**Ponte a prueba 1**

**Desarrollo multiplataforma**

En la parte inicial de la unidad hemos visto que tanto el lenguaje de programación Python como el framework Qt son ampliamente utilizados en el desarrollo de aplicaciones multiplataforma en diferentes campos.

La característica de ser multiplataforma permite que nuestros desarrollos se ejecuten tanto en Windows como en Linux y en macOS.

**¿Cómo podemos probar nuestras aplicaciones de forma cómoda en estas plataformas sin cambiar de máquina?**

1. De las siguientes aplicaciones, ¿cuáles están desarrolladas utilizando Qt? Indica varias respuestas.

a) VirtualBox

b) LibreOffice

c) Microsoft Office

d) Windows

e) Escritorio KDE

2. Python es un lenguaje de programación nuevo que se utiliza relativamente poco, pero tiene gran proyección.

a) Verdadero

b) Falso

3. Python es un lenguaje de programación

a) Compilado

b) Interpretado

4. Una de las ventajas de utilizar Python es que se trata de un lenguaje de programación “ “, lo que nos permite poder utilizar la programación estructurada, la orientación a objetos o la “ “. Otra ventaja es que se trata de un lenguaje

“ “, lo que nos permite ejecutar el código en diferentes plataformas. (modularidad, multiparadigma y multiplataforma)

5. La licencia LGPL (GNU Lesser General Public License) de Qt es apropiada para software propietario.

a) Verdadero

b) Falso

6. Necesitamos comprar una licencia comercial para publicar nuestro proyecto open-source.

a) Verdadero

b) Falso

7. PySide es la unión de Python y Qt, lo que nos permite utilizar Qt desde Python.

a) Verdadero

b) Falso

8. PySide, PySide2 y PySide 6 son compatibles con Qt6

a) Verdadero

b) Falso

9. Python es un lenguaje de programación

a) Libre

b) Propietario

10. Python es un lenguaje de programación

a) alto nivel

b) bajo nivel

Ejercicios Tema 2.2

Caso práctico 2

**QLineEdit y QLabel**

Desarrolla una aplicación con una ventana con un QLineEdit y un QLabel. Asigna un tamaño máximo de texto de 5 carácteres al QLineEdit y un tamaño fijo de 50x30 píxeles. El QLabel también tendrá un tamaño fijo de 50x30 píxeles y se desplazará 50 píxeles a la derecha para no solaparse con el QLineEdit. Cuando el texto del QLineEdit cambie, la etiqueta mostrará el texto introducido.

Solución: Instrucciones adicionales sobre su uso en un contexto práctico real.

Si lo ejecutas, podrás constatar que el cambio en la línea de texto hace que el texto en la etiqueta cambie automáticamente.

Fíjate también en que hemos utilizado los métodos setFixedSize para definir un tamaño fijo, y el método move para moverlo dentro de la interfaz. En el apartado siguiente veremos cómo gestionar esto de forma más cómoda con layouts*.*

En cuanto a las conexiones entre señales y ranuras, la señal textChanged pasa el valor del texto contenido en el QLineEdit cuando se emite a la ranura setText, que lo recibe como argumento.

En la documentación de cada componente se puede consultar si sus señales y ranuras pasan y reciben argumentos, respectivamente, y de qué tipo son.

Puedes repetir este ejercicio cambiando el componente QLineEdit por cualquier otro de entrada de datos (QDateTimeEdit, QDial, QSlider, QComboBox...) y reflejar su contenido en el QLAbel.

**Ponte a prueba 2**

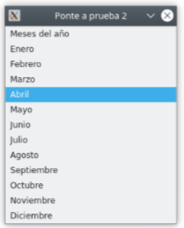
**QComboBox**

Uno de los widgetsque hemos mencionado en la teoría es el QComboBox, que permite seleccionar un elemento de una lista desplegable. En este

Ponte a prueba se pretende que rellenes un combobox con los meses del año; al seleccionar uno de los elementos, se imprimirá por la consola el número que ocupa en el combobox y el texto.

En la imagen puedes ver un ejemplo.

Abril es el mes número 4.



Solución:

- Para rellenar el QComboBox, puedes definir una lista con los meses y pasársela a addItems().

-Si consultas la documentación de QComboBox, verás que al cambiar la selección puede emitir diferentes señales. Hay dos que nos podrían interesar:

* currentIndexChanged(int index) pasa un entero que representa la posición dentro del combobox seleccionado.
* currentTextChanged(const QString &text)) pasa el texto que contiene el elemento seleccionado.

-Implementa una ranura que escuche una de las dos señales e imprima por consola el texto deseado. Se pueden utilizar los parámetros emitidos por la señal o los métodos que nos devuelven la información del elemento seleccionado:

* currentText()
* currentIndex()

1. La reutilización de componentes no es una buena práctica en el desarrollo de software.

a) Verdadero

b) Falso

2. Una aplicación está compuesta por un solo componente.

a) Verdadero

b) Falso

3. Los componentes de una aplicación se estructuran en forma de árbol jerárquico.

a) Verdadero

b) Falso

4. Para que una etiqueta sea visible:

a) Necesariamente ejecutamos su método show.

b) Creamos un objeto de la clase QLabel y será visible por defecto.

c) Le pasamos un parent en su construcción y entonces se hará visible al hacer visible al parent.

d) Le quitamos la propiedad hidden.

5. ¿Cuáles de los siguientes Widgets son componentes de Qt6?

a) QLabel

b) QText

c)QDataBase

d) QDial

e) QSlider

5. Un clic de ratón sobre un botón o la pulsación de una combinación de teclas son que se añadirán a la. El comprobará en cada iteración si hay pendientes de ser gestionados. Eventos, cola de eventos y bucle de eventos



6. Una en Qt se emite cuando el usuario produce un



. Las son funciones que se ejecutarán al lanzarse la emisión de la a la que están conectados.



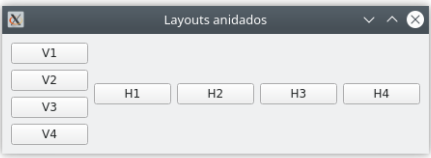
Evento, señal y ranuras

Ejercicios Tema 2.3

**Caso práctico 3**

**Layouts anidados**

Los layouts no solamente pueden contener widgets, sino también otros layouts que, a su vez, pueden contener widgets y layouts. Para añadir un layout como layout hijo, usaremos el método addLayout del padre, y le pasaremos el layout hijo como argumento. Desarrolla una aplicación que tenga el aspecto de la imagen.



Ejercicio DI\_U02\_A03\_CP\_01

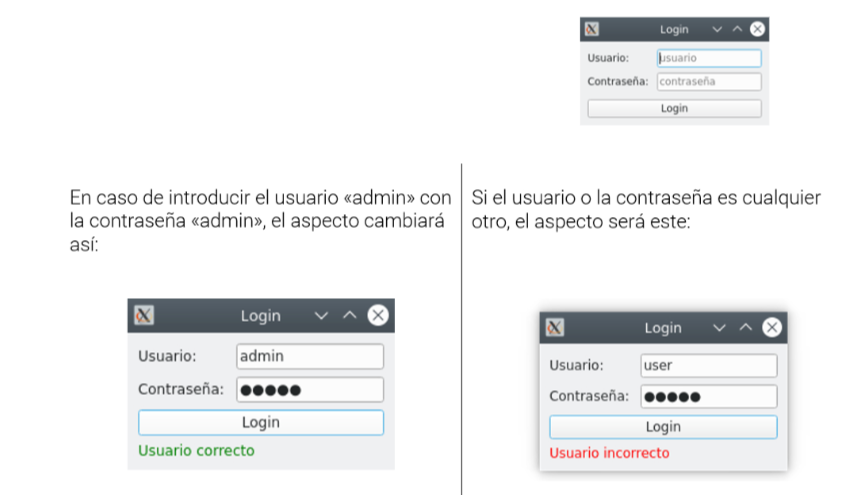
Fíjate en que tenemos un layout principal en horizontal y a este le añadimos dos layouts, uno vertical y otro horizontal, con cuatro botones cada uno.

En adelante, puedes analizar cómo están construidas a nivel de layouts las aplicaciones que utilizas habitualmente. Esto te permitirá tomar como punto de partida de tus desarrollos otras aplicaciones ya diseñadas.

**Ponte a prueba 3**

**Login**

Vamos a crear una aplicación que simule una ventana de login con el aspecto de la imagen.



Solución: **Claves de resolución**

Analiza el layout principal: de qué tipo es y qué layouts hijo y qué widgetsincorpora.

Un widget se puede ocultar o mostrar con el método setHidden(True) o setHidden(False). Además, su aspecto se puede modificar con setStyleSheet() y pasándole reglas CSS como argumento.

Se pueden usar funciones lambda de Python para evitar el uso de propiedades públicas en la ranura. Aunque no es obligatorio, sería recomendable.

Ten en cuenta que poner los usuarios y las contraseñas en el código no es una buena práctica, ya que para cambiarlos sería necesario editar el código. Además, supone un gran riesgo de seguridad, porque se obtendrían las credenciales al hacer ingeniería inversa y obtener el código original de la aplicación. En las aplicaciones reales se consultaría un servidor de bases de datos o algún archivo protegido.

1. Un layout puede contener widgets.

a) Verdadero.

B ) Falso.

2. A un widget solo se le puede asignar un tipo de layout.

a) Verdadero.

B )Falso.

3. Indica el tipo de layout que utilizarías para desarrollar el teclado de una calculadora:

a) Un QVBoxLayout

b) Un QHBoxLayout

c) Un QGridLayout

d) Un QFormLayout

4. Indica el tipo de layout que utilizarías para desarrollar un formulario de alta de usuario:

a) Un QGridLayout

b) Un QFormLayout

c) La combinación de QVBoxLayout y QHBoxLayout

d) Un QStackedLayout

5. Queremos implementar una aplicación que muestre una zona geográfica en concreto y con botones poder cambiar la vista entre «satélite», «mapa de carreteras», «mapa topográfico»... ¿Qué tipo de layout utilizarías?

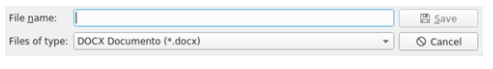
a) QGridLayout

b) QFormLayout

c) QStackedLayout

d) QVerticalLayout

6. Indica los layouts utilizados en la imagen:



a) Un QVBoxLayout como principal, un QFormLayout y un QHBoxLayout

b) Un QFormLayout y un QVBoxLayout

c) Un QHBoxLayout como principal, un QFormLayout y un QVBoxLayout

d) Un QGridLayout

2.4 Ejercicios

Caso práctico 4

What's This?

La ayuda «What's This?» es parte del sistema de ayuda en línea de una aplicación y brinda a los usuarios información sobre la funcionalidad y el uso de un widget en particular.

WhatsThis? proporciona una sola ventana con un texto explicativo que aparece cuando el usuario pincha sobre «¿Qué es esto?» y de nuevo sobre cualquier otro widget. La forma predeterminada para que los usuarios hagan la pregunta es mover el foco al widget relevante y presionar Shift+F1. El texto de ayuda aparece inmediatamente; desaparece tan pronto como el usuario hace otra cosa.

Para entrar en el modo ayuda necesitamos llamar al método estático (no es necesario declarar un objeto de la clase) enterWhatsThisMode(), mientras que para salir de este se llamará a leaveWhatsThisMode(). Se puede consultar en cualquier momento en qué modo se está llamando a inWhatsThisMode().

What's This?

Ahora creamos una aplicación con un componente tipo dock con un QTextEdit a la aplicación y un componente principal. Por defecto, el dock se situará en la parte superior de la ventana.

Añadimos una acción «Imprimir en dock» que imprima en un dock «Acción Pulsada». Su atajo será Ctrl + P y, además, aparecerá en una barra de herramientas y en un menú. Su texto de ayuda será el siguiente: «Al ejecutar esta acción, se añadirá el texto "Acción pulsada" en el dock. Se puede lanzar por *Menú* > *Imprimir en dock*, con Ctrl + P o haciendo clic en el botón correspondiente de la barra de herramientas».

Añadimos un botón *¿Qué es esto?* a la aplicación con el comportamiento habitual.



Ponte a prueba 4

Editor de textos

Vamos a crear un editor de texto plano muy simple con el aspecto que ves en la imagen.

Tendrá tres acciones:

* Abrir archivo, en el menú y en la barra de herramientas. Atajo: Ctrl + o
* Guardar archivo, en el menú y en la barra de herramientas. Atajo: Ctrl + s
* Salir, en el menú. Atajo: Ctrl + q

El archivo en donde guarda o que puede cargar siempre será el mismo, archivo.txt, y estará situado en la misma ruta que el ejecutable. Al guardar, el archivo se sobrescribirá.

Solucion:Claves de resolución

Define tres QAction para las acciones. Cada una irá conectada a una ranura con la funcionalidad especificada y la situarás en la barra de herramientas o en el menú, según corresponda.

Para leer o escribir en un archivo, utiliza el método de Python «open» en el modo que corresponda:

· «r+» para lectura y actualización.

· «w» para escritura.

Para salir de la aplicación, recuerda que normalmente las aplicaciones terminan cuando se cierra la última de sus ventanas visibles.

1. Indica cuáles de los elementos siguientes suelen forman parte de una aplicación de escritorio:

a)Menú.

b) Barra de progreso.

c) Barra de herramientas.

d) Barra de estado.

e) Componente de carga.

2. En las aplicaciones se puede ejecutar una misma” “ interaccionando con diferentes interfaces de usuario, ya sea a través de “ “botones de la “” o” “ Aquí es donde entran en juego las “ “ de Qt. Además, se les puede asignar un texto de estado, que será usado en la barra de “ “y un texto de ayuda, que se mostrará al entrar en el modo “ “.

3. Las QAction no permiten reutilizar código, ya que, si queremos que un botón de la barra de herramientas, una entrada en el menú o un atajo de teclado realicen la misma función, debemos definirla tres veces, con lo que es más difícil el mantenimiento de la aplicación.

a) Verdadero.

b) Falso.

4. Si queremos asegurarnos de mostrar el texto debajo de los iconos en las barras de herramientas, debemos pasarle a setButtonStyle el parámetro:

a) Qt.ToolButtonIconOnly

b) Qt.ToolButtonTextOnly

c) Qt.ToolButtonTextBesideIcon

d) Qt.ToolButtonTextUnderIcon

e) Qt.ToolButtonFollowStyle

EJERCICIO 2.5

Caso práctico 5

Abrir y guardar

En el Ponte a prueba del apartado anterior empezamos a desarrollar un editor de texto muy simple que permita cargar y guardar el «archivo.txt» situado en la misma ruta desde la que ejecutábamos el código.

En este caso práctico, vamos a utilizar dos diálogos: uno para pedir qué archivo abrir y otro para pedir en qué archivo queremos guardar los cambios. En caso de que ya haya un archivo abierto, al hacer clic en *Guardar* no se pedirá la ruta al archivo, sino que se utilizará la ruta del archivo abierto anteriormente, sobrescribiéndolo.

Además, añadiremos una entrada de menú *Cerrar* para cerrar el archivo abierto actualmente y abrir uno nuevo.

Ejercicio 5.Ten en cuenta que, al salir de la aplicación, cerrar el archivo o abrir otro archivo, no comprobamos si hemos guardado los cambios ni preguntamos al usuario si desea guardarlos, con lo que estos se pierden. Dejamos como ampliación esta funcionalidad.

Ponte a prueba 5

Diálogo

En el Apartado 3 habíamos creado una ventana para pedir el usuario y la contraseña. Vamos a cambiar su implementación haciendo que sea un diálogo, de forma que si el usuario y la contraseña son «admin»/ «admin» entraremos en la aplicación, que mostraremos de forma maximizada. En caso contrario, mostraremos un cuadro de diálogo nuevo indicando que «El usuario o la contraseña son incorrectos».

La ventana de la aplicación simplemente contendrá un QLabel con el texto «Ventana principal».

SOl:El diálogo se mostrará al lanzar su bucle de eventos, función exec(). Esta función nos devolverá el resultado del diálogo, así que podemos utilizar la ranura accept() cuando el login sea correcto, lo que nos devolverá el valor QDialog.Accepted.

En caso de devolver otra cosa, mostraremos el mensaje de login incorrecto indicado en el enunciado utilizando un QMessageBox.