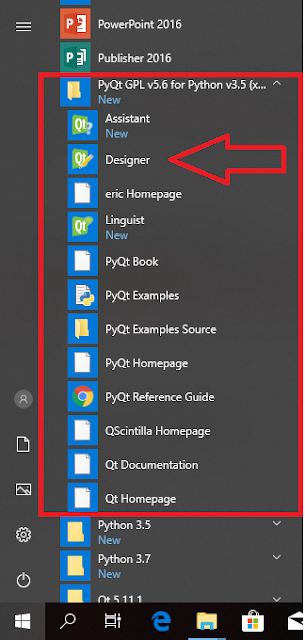
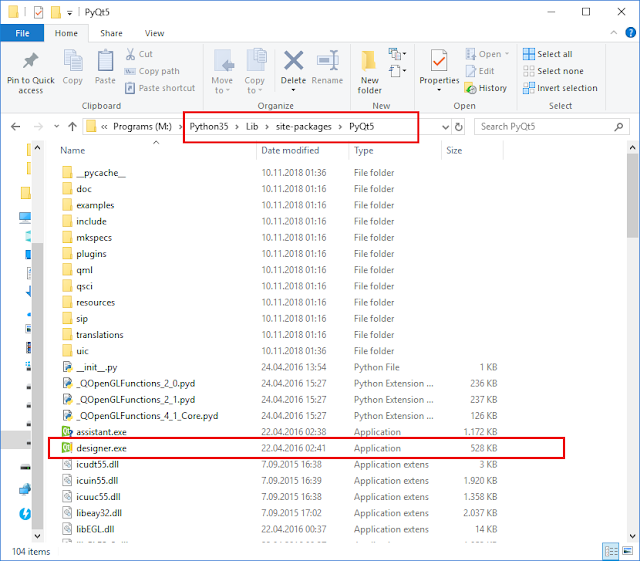
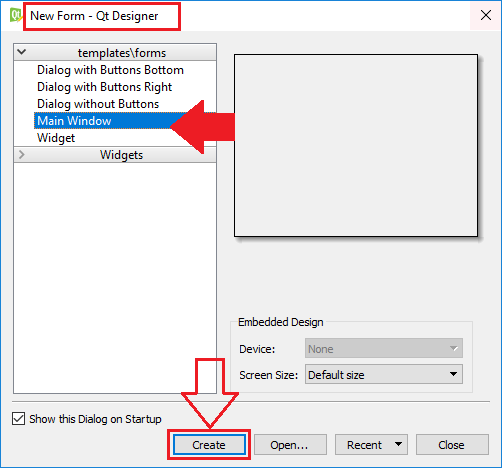
Diseño con Qt Designer

Paso 1.1.  
Se abre Qt Designer. Puede encontrar Qt Designer en la carpeta PyQt en el menú Inicio o en la carpeta donde instaló Python ../Python35/Lib/site-packages/PyQt5 .

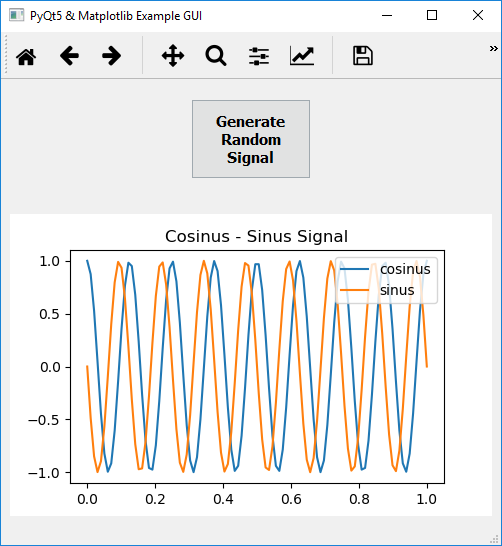
[](https://3.bp.blogspot.com/-8DJ_POeiZDs/W-YeBstCrFI/AAAAAAAABDc/TNI-KkQNNxoEeK3Gn9uGeCk1I1DcBnKLACLcBGAs/s1600/PyQt5_matplotlib_ad%25C4%25B1m_1_1.PNG)

[](https://3.bp.blogspot.com/-UIDyuBBB4JA/W-YeH-jQz4I/AAAAAAAABDg/lvjIjZQPpnkM7Eig_BGaTdGpDR1ghKTTACLcBGAs/s1600/PyQt5_matplotlib_ad%25C4%25B1m_1_2.PNG)

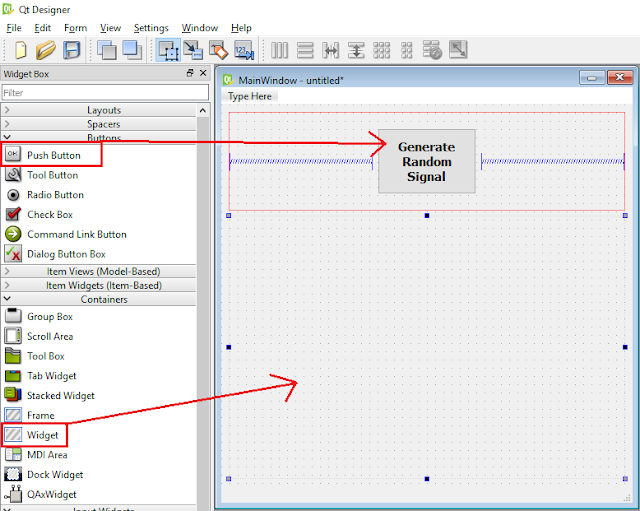
Paso 1.2.  
Cuando se abre Qt Designer, aparece una ventana que le permite abrir un diseño previamente preparado o crear un nuevo diseño. Básicamente, las opciones de Diálogo, Ventana principal y Widget están disponibles aquí. Los diálogos son ventanas que solicitan al usuario que elija una opción o informan al usuario. En la ventana principal , puede crear un menú ( barra de menús ) diferente de la ventana de widgets y presentar mensajes informativos al usuario en la sección llamada barra de estado en la parte inferior de la interfaz . Desde esta ventana, elegimos Main Window para un nuevo diseño . Puedes hacer una elección diferente.

[](https://1.bp.blogspot.com/-8dAyt5IFtA0/W-YgVTw-IdI/AAAAAAAABDw/xUOIsp1uiicrM7kQFo_8mJz19R3ncyp_ACLcBGAs/s1600/PyQt5_matplotlib_ad%25C4%25B1m_2.PNG)

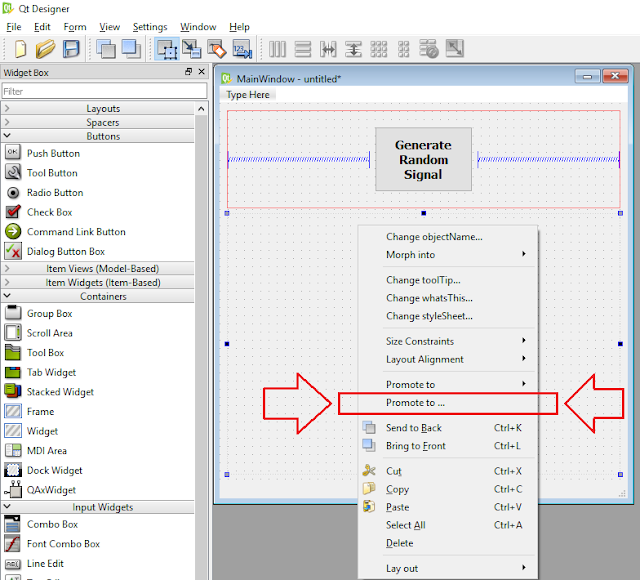
Paso 1.3.  
Puede encontrar la captura de pantalla de la interfaz que queremos diseñar a continuación. Una interfaz simple que consta de un botón y un gráfico, que actualiza el gráfico cuando se presiona el botón. Como es muy difícil describir los detalles del diseño por escrito, pondré un video al final del artículo, pueden verlo desde allí.

[](https://1.bp.blogspot.com/-P7cFnGEM9lc/W-d9Veyu5AI/AAAAAAAABGU/q5_cW-OMmYwKol5ozHOrWhG83gbWhobxgCLcBGAs/s1600/PyQt5_matplotlib_python_2_3.PNG)

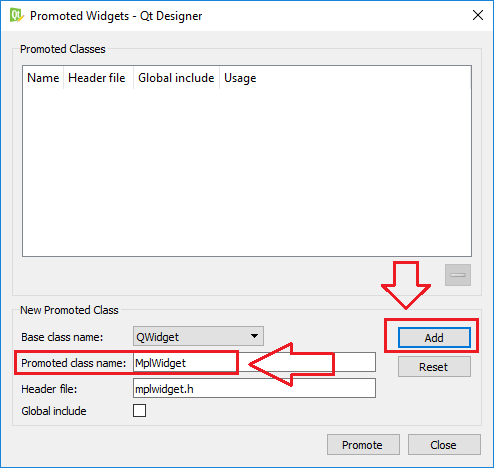
Paso 1.4.  
Los objetos Push Button y Widget se colocan en la interfaz mediante el método de arrastrar y soltar.

[](https://2.bp.blogspot.com/-ceRNXkQCUpk/W-YkcUWa9oI/AAAAAAAABEU/l9buCmou3rAemeX8PD9Dhd_knBvtKrU_ACLcBGAs/s1600/PyQt5_matplotlib_ad%25C4%25B1m_4.PNG)

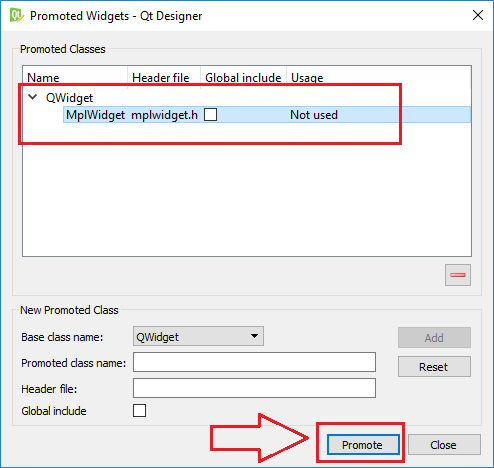
Paso 1.5. Mostraremos los gráficos que produciremos con matplotlib en el  
widget . Para ello, haz clic con el botón derecho en el widget y selecciona Promocionar a... de las opciones que aparecen .

[](https://4.bp.blogspot.com/-WnEL3WOyNmA/W-YluyGPKEI/AAAAAAAABEg/fLfTKgjWseIZr0-74i1qL86dt3jd3VkgwCLcBGAs/s1600/PyQt5_matplotlib_ad%25C4%25B1m_5.PNG)

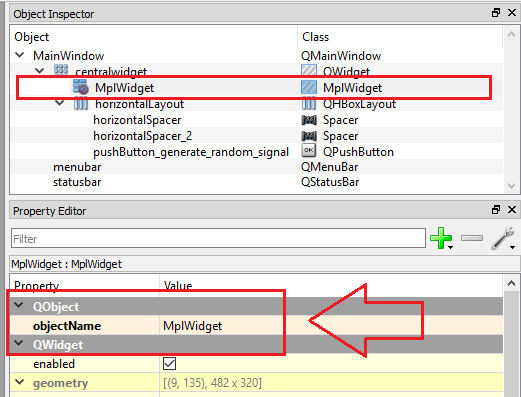
Paso 1.6.  
Después de seleccionar Promocionar a... se abrirá una nueva ventana como se muestra en la siguiente captura de pantalla. En esta ventana , MplWidget se escribe usando el prefijo Mpl, que es la abreviatura de matplotlib, en la sección que dice Nombre de clase promocionada :. Puedes darle el nombre que quieras. Cuando lo escriba aquí, la sección Archivo de encabezado: se completará automáticamente de acuerdo con el nombre que escribió. Luego haga clic en el botón Agregar .

[](https://4.bp.blogspot.com/-o52N1NMDY5s/W-YoPB5lpRI/AAAAAAAABEs/zHgEAJYAhFsSJQdP-WwG4ObSTurlBLzpgCLcBGAs/s1600/PyQt5_matplotlib_ad%25C4%25B1m_6_2.PNG)

Después de completar los pasos mencionados anteriormente, haga clic en el botón Promocionar .

[](https://1.bp.blogspot.com/-glmWOcIGruw/W-YpRxWbZuI/AAAAAAAABE4/j7xAea_hLKYKdISR2mz9iChOEIzlzIBnwCLcBGAs/s1600/PyQt5_matplotlib_ad%25C4%25B1m_6_3.PNG)

Nota.  
Además de los pasos mencionados anteriormente, el nombre de objeto ( objectName ) del widget en la interfaz (donde se realiza la operación de Promoción) se hace con el mismo nombre que la variable de nombre de clase Promovida .

[](https://4.bp.blogspot.com/-M73CmzYfTLE/W-YqjK0Cv6I/AAAAAAAABFQ/YGOJxsGVCbEfq2SvSuYtE1XxBDPzkffNwCLcBGAs/s1600/PyQt5_matplotlib_ad%25C4%25B1m_6_4.PNG)

Paso 1.7.  
Finalmente, guardamos nuestro diseño como .ui con la combinación de teclas Ctrl + S. Con este proceso, se terminan las cosas por hacer en la fase de diseño. Ahora podemos pasar a la etapa de usar la interfaz que diseñamos con los códigos de Python.

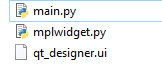
2. Usando Qt Designer con Python

Paso 2.1.  
Habrá 3 archivos en total en nuestro proyecto:

main.py: nuestro archivo principal de Python donde cargaremos la interfaz que diseñamos en el diseñador Qt y realizaremos operaciones en ella.

mplwidget.py: nuestro archivo de Python que tiene el mismo nombre que el widget que se promociona en el diseñador Qt. Estas son las operaciones que permiten usar matplotlib en el widget.

qt\_designer.ui: archivo de interfaz diseñado en Qt designer.

[](https://3.bp.blogspot.com/-WfBFJsXDA0M/W-cU1rWsoAI/AAAAAAAABFk/GTYDmbMiFHclafBwvJCo6MBei6x0tkMFQCLcBGAs/s1600/PyQt5_matplotlib_python_2_1.PNG)

|  |  |
| --- | --- |
|  |  |

Los códigos anteriores son los códigos en el archivo Python mplwidget.py . Ahora vamos a explicar estos códigos paso a paso.  
Las primeras 3 líneas son líneas de comentarios. Está escrito para indicar a qué archivo pertenecen los códigos.  
En la línea 4 se agregan todos los componentes pertenecientes a la clase QtWidgets.  
FigureCanvas del módulo Matplotlib se agrega a nuestro código en la línea 6.  
Matplotlib consta de 3 capas diferentes y hay Backend en la capa inferior , Artista en la capa superior y Capa de secuencias de comandos en la capa superior. Puede encontrar información detallada sobre la estructura del módulo Matplotlib [aquí](http://www.aosabook.org/en/matplotlib.html) .  
en la capa de fondoFigureCanvas se puede considerar como el papel (lienzo) que necesitamos para dibujar. FigureCanvas es una clase de Matplotlib, así como una clase de QWidget.  
En la capa Artista , el título, gráfico, eje xy, etiquetas, etc. lo vemos en Matplotlib. los componentes están incluidos.  
La capa de script (pyplot) es la capa superior que facilita el uso del módulo Matplotlib. En la línea 8, la clase Figure  
del módulo Matplotlib se agrega a nuestro código.

|  |  |
| --- | --- |
|  |  |

En la línea 11, definimos la clase llamada MplWidget con el mismo nombre que el Widget que producimos a partir de la clase QWidget en la línea y que hicimos la operación Promover en Qt Designer . Cada clase tiene una función \_\_init\_\_() , que se ejecuta primero cuando se llama a la clase. La declaración self se usa para indicar que una variable definida en una clase es una función, variable, de modo que las funciones se pueden llamar desde fuera de la clase. Tiene el mismo uso que la declaración this utilizada en el lenguaje de programación C++ .  
La línea 15 llama a la función \_\_init\_\_() de la superclase.

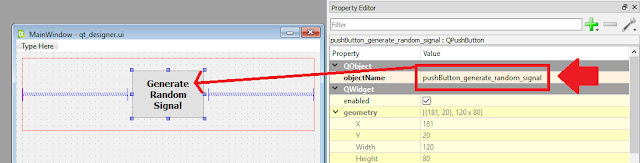
|  |  |
| --- | --- |
|  |  |

En la línea 17, se define un nuevo objeto de la clase FigureCanvas y el nombre del objeto se define como "canvas" y este widget se asigna a la declaración propia ya que necesitaremos acceder a él fuera de la clase (en el archivo main.py) .  
Definimos un diseño vertical ( QVBoxLayout ) en la línea 19 y agregamos el objeto FigureCanvas que creamos usando la función addWidget en la línea 20 en este diseño vertical. En PyQt, sus diseños tienen un diseño horizontal (QHBoxLayout) y un diseño de cuadrícula (QGridLayout), a diferencia del diseño vertical (QVBoxLayout). [Definimos nuevos ejes](https://matplotlib.org/api/axes_api.html#matplotlib.axes.Axes)  
para el lienzo (FigureCanvas) que creamos en la línea 22 . [add\_subplot](https://matplotlib.org/api/_as_gen/matplotlib.figure.Figure.html?highlight=add_subplot" \l "matplotlib.figure.Figure.add_subplot) para esta acciónUsamos la función.  
Definimos el diseño vertical que creamos en la línea 23 con la función setLayout para asegurarnos de que nuestra clase (la interfaz que crearemos) sea el diseño. Con esta línea finaliza la descripción de los códigos mplwidget.py .

Paso 2.2.  
Los códigos y explicaciones en el archivo de Python main.py se dan a continuación.

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | *# ------------------------------------------------------*  *# ---------------------- main.py -----------------------*  *# ------------------------------------------------------*  **from** PyQt5.QtWidgets **import\***  **from** PyQt5.uic **import** loadUi  **from** matplotlib.backends.backend\_qt5agg **import** **(**NavigationToolbar2QT **as** NavigationToolbar**)**  **import** numpy **as** np  **import** random    **class** MatplotlibWidget**(**QMainWindow**):**    **def** \_\_init\_\_**(**self**):**    QMainWindow**.**\_\_init\_\_**(**self**)**  loadUi**(**"qt\_designer.ui"**,**self**)**  self**.**setWindowTitle**(**"PyQt5 & Matplotlib Example GUI"**)**  self**.**pushButton\_generate\_random\_signal**.**clicked**.**connect**(**self**.**update\_graph**)**  self**.**addToolBar**(**NavigationToolbar**(**self**.**MplWidget**.**canvas**,** self**))**  **def** update\_graph**(**self**):**  fs **=** **500**  f **=** random**.**randint**(1,** **100)**  ts **=** **1/**fs  length\_of\_signal **=** **100**  t **=** np**.**linspace**(0,1,**length\_of\_signal**)**    cosinus\_signal **=** np**.**cos**(2\***np**.**pi**\***f**\***t**)**  sinus\_signal **=** np**.**sin**(2\***np**.**pi**\***f**\***t**)**  self**.**MplWidget**.**canvas**.**axes**.**clear**()**  self**.**MplWidget**.**canvas**.**axes**.**plot**(**t**,** cosinus\_signal**)**  self**.**MplWidget**.**canvas**.**axes**.**plot**(**t**,** sinus\_signal**)**  self**.**MplWidget**.**canvas**.**axes**.**legend**((**'cosinus'**,** 'sinus'**),**loc**=**'upper right'**)**  self**.**MplWidget**.**canvas**.**axes**.**set\_title**(**'Cosinus - Sinus Signal'**)**  self**.**MplWidget**.**canvas**.**draw**()**    app **=** QApplication**([])**  window **=** MatplotlibWidget**()**  window**.**show**()**  app**.**exec\_**()** | |

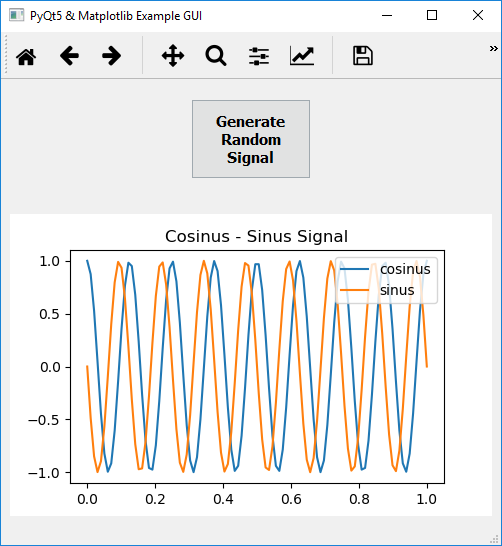
Se ha agregado la función loadUi en el módulo uic en la línea 5 y esta función se usa para cargar el diseño que hemos hecho con Qt designer con extensión .ui en la línea 18. NavigationToolBar2QT  
del módulo Matplotlib se incluye en el código para poder operar en el gráfico de la línea 7. En la línea 9, se agregó el paquete NumPy para su uso en operaciones matemáticas y operaciones de matrices. En la línea 10, el paquete aleatorio que viene con la instalación de Python se agregó al código para generar números aleatorios.  
  
  
Expliqué las líneas 12 a 16 mientras describía los códigos mplwidget.py, no los explicaré nuevamente aquí. El punto a tener en cuenta es que define una clase de acuerdo con la estructura que diseña con Qt Designer. Diseñamos seleccionando Main Window en Qt Designer , así que aquí usamos QMainWindow en la definición de clase .  
En la línea 18 se carga el diseño realizado con Qt Designer. Debe dar el nombre con el que guardó el diseño como primer parámetro.  
En la línea 20, nombramos la ventana de interfaz con la función setWindowTitle . Conectamos la función update\_graph (slot) a la señal de clic  
del botón que definimos en Qt Designer en la línea 22 . Para conexión de ranura de señal[Puedes revisar mi publicación anterior](https://yapayzekalabs.blogspot.com/2018/09/pyqt5-ile-grafiksel-kullanc-arayuzu-gui.html) .

[](https://3.bp.blogspot.com/-MH9lZty7LcA/W-dLGwkLmJI/AAAAAAAABFw/1pY2x9PAmTk28cO7aE0-eeKtvwRQq9aYACLcBGAs/s1600/PyQt5_matplotlib_python_2_2.PNG)

En la línea 24, NavigationToolbar se agrega a la interfaz con la función addToolBar .

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | **def** update\_graph**(**self**):**  fs **=** **500**  f **=** random**.**randint**(1,** **100)**  ts **=** **1/**fs  length\_of\_signal **=** **100**  t **=** np**.**linspace**(0,1,**length\_of\_signal**)**    cosinus\_signal **=** np**.**cos**(2\***np**.**pi**\***f**\***t**)**  sinus\_signal **=** np**.**sin**(2\***np**.**pi**\***f**\***t**)**  self**.**MplWidget**.**canvas**.**axes**.**clear**()**  self**.**MplWidget**.**canvas**.**axes**.**plot**(**t**,** cosinus\_signal**)**  self**.**MplWidget**.**canvas**.**axes**.**plot**(**t**,** sinus\_signal**)**  self**.**MplWidget**.**canvas**.**axes**.**legend**((**'cosinus'**,** 'sinus'**),**loc**=**'upper right'**)**  self**.**MplWidget**.**canvas**.**axes**.**set\_title**(**'Cosinus - Sinus Signal'**)**  self**.**MplWidget**.**canvas**.**draw**()**    app **=** QApplication**([])**  window **=** MatplotlibWidget**()**  window**.**show**()**  app**.**exec\_**()** | |

Se ha escrito la función update\_graph (ranura) conectada al botón entre las líneas 27-43 . Nuestro código ejecutará esta función cada vez que el usuario presione el botón.  
Al usar las funciones en los paquetes NumPy y aleatorios entre las líneas 29-36, se generan funciones de seno y coseno en frecuencias aleatorias.  
Antes de realizar un nuevo dibujo en el gráfico de la línea 38, se eliminan los dibujos anteriores. Si no hacemos esto, dibujará los gráficos uno encima del otro.  
Se grafican las funciones coseno y seno producidas en las líneas 39-40.  
Se añade leyenda para distinguir los gráficos que hemos dibujado en la línea 41.  
Se agrega un título al gráfico en la línea 42.  
En la línea 43, se dibuja nuestro gráfico.  
Las líneas 46-49 están [en mi artículo anterior .](https://yapayzekalabs.blogspot.com/2018/09/pyqt5-ile-grafiksel-kullanc-arayuzu-gui.html)No lo estoy repitiendo aquí porque lo expliqué.

[](https://4.bp.blogspot.com/-FOt-k-F9Ibk/W-dTUHrBe5I/AAAAAAAABF8/ajdXw2mMnpcPbGhCvE4XqciwI0mNtGjaACLcBGAs/s1600/PyQt5_matplotlib_python_2_3.PNG)

Aprendimos a dibujar gráficos usando el paquete Matplotlib en la interfaz diseñada con Qt Designer. Puede haber muchas características que no hemos mencionado, pero puede tomar esto como un comienzo y preparar interfaces mejores y más hermosas para su propio trabajo.