```python
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
##################################################
# GNU Radio Python Flow Graph
# Title: Onda Seno
# GNU Radio version: 3.7.13.5
##################################################

from distutils.version import StrictVersion

if __name__ == '__main__':
    import ctypes
    import sys
    if sys.platform.startswith('linux'):
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print "Warning: failed to XInitThreads()"

from PyQt5 import Qt
from PyQt5 import Qt, QtCore
from gnuradio import analog
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio import qtgui
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.qtgui import Range, RangeWidget
from optparse import OptionParser
import sip
import sys
from gnuradio import qtgui


class onda_seno(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "Onda Seno")
        Qt.QWidget.__init__(self)
        self.setWindowTitle("Onda Seno")
        qtgui.util.check_set_qss()
        try:
            self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
        except:
            pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)

        self.settings = Qt.QSettings("GNU Radio", "onda_seno")
        self.restoreGeometry(self.settings.value("geometry", type=QtCore.QByteArray))


        ##################################################
        # Variables
        ##################################################
        self.samp_rate = samp_rate = 32e3
        self.my_var = my_var = 0
        self.frec = frec = 1e3

        ##################################################
        # Blocks
        ##################################################
        self._frec_range = Range(0, 16e3, 1000, 1e3, 200)
        self._frec_win = RangeWidget(self._frec_range, self.set_frec, 'frec', "counter_slider", float)
        self.top_grid_layout.addWidget(self._frec_win)
        self.qtgui_time_sink_x_0 = qtgui.time_sink_f(
          1024, #size
          samp_rate, #samp_rate
          "", #name
          1 #number of inputs
        )
        self.qtgui_time_sink_x_0.set_update_time(0.10)
        self.qtgui_time_sink_x_0.set_y_axis(-1, 1)

        self.qtgui_time_sink_x_0.set_y_label('Amplitude', '1')

        self.qtgui_time_sink_x_0.enable_tags(-1, True)
        self.qtgui_time_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_FREE, qtgui.TRIG_SLOPE_POS, 0.0, 0, 0, "")
        self.qtgui_time_sink_x_0.enable_autoscale(True)
        self.qtgui_time_sink_x_0.enable_grid(True)
        self.qtgui_time_sink_x_0.enable_axis_labels(True)
        self.qtgui_time_sink_x_0.enable_control_panel(False)
        self.qtgui_time_sink_x_0.enable_stem_plot(False)

        if not True:
          self.qtgui_time_sink_x_0.disable_legend()
        labels = ['', '', '', '', '',
                '', '', '', '', '']
        widths = [1, 1, 1, 1, 1,
                1, 1, 1, 1, 1]
        colors = ["blue", "red", "green", "black", "cyan",
                "magenta", "yellow", "dark red", "dark green", "blue"]
        styles = [1, 1, 1, 1, 1,
                1, 1, 1, 1, 1]
        markers = [-1, -1, -1, -1, -1,
                -1, -1, -1, -1, -1]
        alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
                1.0, 1.0, 1.0, 1.0, 1.0]

        for i in xrange(1):
            if len(labels[i]) == 0:
                self.qtgui_time_sink_x_0.set_line_label(i, "Data {0}".format(i))
            else:
                self.qtgui_time_sink_x_0.set_line_label(i, labels[i])
            self.qtgui_time_sink_x_0.set_line_width(i, widths[i])
            self.qtgui_time_sink_x_0.set_line_color(i, colors[i])
            self.qtgui_time_sink_x_0.set_line_style(i, styles[i])
            self.qtgui_time_sink_x_0.set_line_marker(i, markers[i])
            self.qtgui_time_sink_x_0.set_line_alpha(i, alphas[i])

        self._qtgui_time_sink_x_0_win = sip.wrapinstance(self.qtgui_time_sink_x_0.pyqwidget(), Qt.QWidget)
        self.top_grid_layout.addWidget(self._qtgui_time_sink_x_0_win)
        self.blocks_throttle_0 = blocks.throttle(gr.sizeof_float*1, samp_rate,True)
        self.analog_sig_source_x_0 = analog.sig_source_f(samp_rate, analog.GR_COS_WAVE, frec, 1, 0)



        ##################################################
        # Connections
        ##################################################
        self.connect((self.analog_sig_source_x_0, 0), (self.blocks_throttle_0, 0))
        self.connect((self.blocks_throttle_0, 0), (self.qtgui_time_sink_x_0, 0))

    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio", "onda_seno")
        self.settings.setValue("geometry", self.saveGeometry())
        event.accept()

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.qtgui_time_sink_x_0.set_samp_rate(self.samp_rate)
        self.blocks_throttle_0.set_sample_rate(self.samp_rate)
        self.analog_sig_source_x_0.set_sampling_freq(self.samp_rate)

    def get_my_var(self):
        return self.my_var

    def set_my_var(self, my_var):
        self.my_var = my_var

    def get_frec(self):
        return self.frec

    def set_frec(self, frec):
        self.frec = frec
        self.analog_sig_source_x_0.set_frequency(self.frec)


def main(top_block_cls=onda_seno, options=None):

    qapp = Qt.QApplication(sys.argv)

    tb = top_block_cls()
    tb.start()
    tb.show()

    def quitting():
        tb.stop()
        tb.wait()
    qapp.aboutToQuit.connect(quitting)
    qapp.exec_()


if __name__ == '__main__':
    main()
```