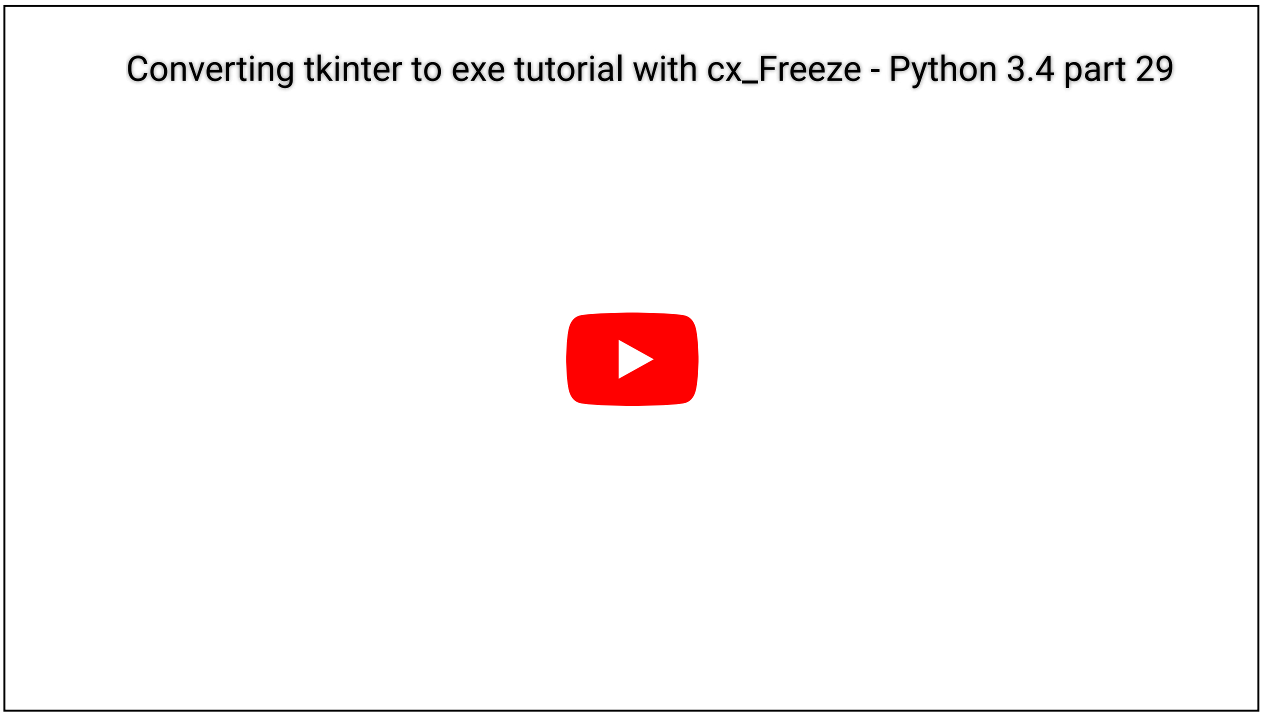
Converting Tkinter application to .exe and installer with cx_Freeze


Converting tkinter to exe tutorial with cx_Freeze - Python 3.4 part 29

setup.py code:

```
import cx_Freeze
import sys
import matplotlib

base = None

if sys.platform == 'win32':
    base = "Win32GUI"

executables = [cx_Freeze.Executable("tkinterVid28.py", base=base, icon="cli
    options = {"build_exe": {"packages":["tkinter","matplotlib"], "include_
    version = "0.01",
    description = "Sea of BTC trading application",
    executables = executables
    )
```

The icon if you don't have it:

Saved as tkinterVid28.py:

```
# The code for changing pages was derived from: http://stackoverflow.com/qu
# License: http://creativecommons.org/licenses/by-sa/3.0/

import matplotlib
matplotlib.use('TkAgg')
import matplotlib.animation as animation
from numpy import arange, sin, pi
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, Navigation
from matplotlib.figure import Figure
import matplotlib.dates as mdates
import matplotlib.ticker as mticker
from matplotlib.finance import candlestick_ohlc

import tkinter as tk
from tkinter import ttk
from matplotlib import style

import urllib
import json
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt



style.use('ggplot')
```

```
LARGE_FONT= ("Verdana", 12)
NORM_FONT = ("Helvetica", 10)
SMALL_FONT = ("Helvetica", 8)


f = plt.figure()
a = f.add_subplot(111)

exchange = "BTC-e"
DatCounter = 9000
programName = "btce"

resampleSize = "15Min"




#########################
## set this to 1day.
DataPace = "1d"
########################
paneCount = 1
candleWidth = 0.008

topIndicator = "none"
bottomIndicator = "none"
middleIndicator = "none"
chartLoad = True

EMAs = []
SMAs = []


darkColor = '#183A54'
lightColor = '#00A3E0'




def tutorial():
    def leavemini(what):
```

```
def page2():
    leavemini(tut)
    tut2 = tk.Tk()
    def leavemini2(what):
        what.destroy()

    def page3():

        leavemini2(tut2)
        tut3 = tk.Tk()
        tut3.wm_title("part 3!")

        label = ttk.Label(tut3, text="Part 3", font=NORM_FONT)
        label.pack(side="top", fill="x", pady=10)
        B1 = ttk.Button(tut3, text = "Done!", command = tut3.destroy)
        B1.pack()
        tut3.mainloop()



    tut2.wm_title("part 2!")

    label = ttk.Label(tut2, text="Part 2", font=NORM_FONT)
    label.pack(side="top", fill="x", pady=10)
    B1 = ttk.Button(tut2, text = "next!", command = page3)
    B1.pack()

    tut.mainloop()


tut = tk.Tk()
tut.wm_title("Tutorial")
label = ttk.Label(tut, text="What do you need help with?", font=NORM_FO
label.pack(side="top", fill="x", pady=10)
B1 = ttk.Button(tut, text = "Overview of the application", command = pa
B1.pack()

B2 = ttk.Button(tut, text = "How do I trade here?", command=lambda: pop
B2.pack()

B3 = ttk.Button(tut, text = "Indicator questions/help", command=lambda:
B3.pack()
```

```
def loadChart(run):
    global chartLoad

    if run == 'start':
        chartLoad = True
    elif run == 'stop':
        chartLoad = False
```

```
def addTopIndicator(what):
    global topIndicator
    global DatCounter

    if DataPace == "tick":
        popupmsg("Indicators in Tick Data not available, choose 1 minute tf

    if what == "none":
        topIndicator = what
        DatCounter = 9000

    elif what == "rsi":
        rsiQ = tk.Tk()
        rsiQ.wm_title("Periods?")
        label = ttk.Label(rsiQ, text="Choose how many periods you want each
        label.pack(side="top", fill="x", pady=10)

        e = ttk.Entry(rsiQ)
        e.insert(0,14)
        e.pack()
        e.focus_set()

        def callback():
            periods = (e.get())
            group = []
            group.append("rsi")
            group.append(periods)
            topIndicator = group
            DatCounter = 9000
            print("set top indicator to",group)
            rsiQ.destroy()

        b = ttk.Button(rsiQ, text="Submit", width=10, command=callback)
        b.pack()

        tk.mainloop()

    elif what == "macd":
        global topIndicator
        global DatCounter
        topIndicator = "macd"
        DatCounter = 9000




def addMiddleIndicator(what):
    global middleIndicator
    global DatCounter
    if DataPace == "tick":
        popupmsg("Indicators in Tick Data not available, choose 1 minute tf

    if what != "none":
        if middleIndicator == "none":

            if what == "sma":
                midIQ = tk.Tk()
                midIQ.wm_title("Periods?")
                label = ttk.Label(midIQ, text="Choose how many periods you
                label.pack(side="top", fill="x", pady=10)
                e = ttk.Entry(midIQ)
                e.insert(0,10)
                e.pack()
                e.focus_set()
                def callback():
```

```
                middleIndicator = []
                periods = (e.get())
                group = []
                group.append("sma")
                group.append(int(periods))
                middleIndicator.append(group)
                DatCounter = 9000
                print("mid indicator",middleIndicator)
                midIQ.destroy()
            b = ttk.Button(midIQ, text="Submit", width=10, command=call
            b.pack()
            tk.mainloop()

        if what == "ema":
            midIQ = tk.Tk()
            midIQ.wm_title("Periods?")
            label = ttk.Label(midIQ, text="Choose how many periods you
            label.pack(side="top", fill="x", pady=10)
            e = ttk.Entry(midIQ)
            e.insert(0,10)
            e.pack()
            e.focus_set()
            def callback():
                global middleIndicator
                global DatCounter
                middleIndicator = []
                periods = (e.get())
                group = []
                group.append("ema")
                group.append(int(periods))
                middleIndicator.append(group)
                DatCounter = 9000
                print("mid indicator",middleIndicator)
                midIQ.destroy()
            b = ttk.Button(midIQ, text="Submit", width=10, command=call
            b.pack()
            tk.mainloop()


    else:
        if what == "sma":
            midIQ = tk.Tk()
            label.pack(side="top", fill="x", pady=10)
            e = ttk.Entry(midIQ)
            e.insert(0,10)
            e.pack()
            e.focus_set()
            def callback():
                global middleIndicator
                global DatCounter
                periods = (e.get())
                group = []
                group.append("sma")
                group.append(int(periods))
                middleIndicator.append(group)
                DatCounter = 9000
                print("mid indicator",middleIndicator)
                midIQ.destroy()
            b = ttk.Button(midIQ, text="Submit", width=10, command=call
            b.pack()
            tk.mainloop()

        if what == "ema":
            midIQ = tk.Tk()
            midIQ.wm_title("Periods?")
            label = ttk.Label(midIQ, text="Choose how many periods you
            label.pack(side="top", fill="x", pady=10)
            e = ttk.Entry(midIQ)
            e.insert(0,10)
            e.pack()
            e.focus_set()
            def callback():
                global middleIndicator
                global DatCounter
                periods = (e.get())
                group = []
                group.append("ema")
                group.append(int(periods))
                middleIndicator.append(group)
                DatCounter = 9000
                print("mid indicator",middleIndicator)
                midIQ.destroy()
            b = ttk.Button(midIQ, text="Submit", width=10, command=call
            b.pack()
        middleIndicator = "none"
```

```
def addBottomIndicator(what):
    global bottomIndicator
    global DatCounter

    if DataPace == "tick":
        popupmsg("Indicators in Tick Data not available, choose 1 minute tf

    if what == "none":
        bottomIndicator = what
        DatCounter = 9000

    elif what == "rsi":
        rsiQ = tk.Tk()
        rsiQ.wm_title("Periods?")
        label = ttk.Label(rsiQ, text="Choose how many periods you want each
        label.pack(side="top", fill="x", pady=10)

        e = ttk.Entry(rsiQ)
        e.insert(0,14)
        e.pack()
        e.focus_set()

        def callback():
            global bottomIndicator
            global DatCounter
            periods = (e.get())
            group = []
            group.append("rsi")
            group.append(periods)
            bottomIndicator = group
            DatCounter = 9000
            print("set top indicator to",group)
            rsiQ.destroy()

        b = ttk.Button(rsiQ, text="Submit", width=10, command=callback)
        b.pack()
    elif what == "macd":
        global bottomIndicator
        global DatCounter
        bottomIndicator = "macd"
        DatCounter = 9000




def changeTimeFrame(tf):
    global DataPace
    global DatCounter
    if tf == '7d' and resampleSize == '1Min':
        popupmsg("Too much data chosen, choose a smaller data time frame or
    else:
        DataPace = tf
        DatCounter = 9000


def changeSampleSize(size,width):
    global resampleSize
    global DatCounter
    global candleWidth

    if DataPace == '7d' and size == '1Min':
        popupmsg("Too much data chosen, choose a smaller Data Time Frame or

    if DataPace == 'tick':
        popupmsg("You are currently viewing tick data, not OHLC. Choose a l

    else:
        resampleSize = size
        DatCounter = 9000
        candleWidth = width


def popupmsg(msg):
    popup = tk.Tk()
    def leavemini():
       popup.destroy()

    popup.wm_title("!")
    label.pack(side="top", fill="x", pady=10)
    B1 = ttk.Button(popup, text = "Okay", command = leavemini)
    B1.pack()

    popup.mainloop()

def changeExchange(toWhat,pn):
    global exchange
    global DatCounter
    global programName
    exchange = toWhat
    programName = pn
    DatCounter = 9000


def animate(i):
    global refreshRate
    global DatCounter

###################################
###################################
    def computeMACD(x, slow=26, fast=12,location="bottom"):
        """
        compute the MACD (Moving Average Convergence/Divergence) using a fa
        return value is emaslow, emafast, macd which are len(x) arrays
        """
        values = {'key': 1,'prices':x}


        url = "http://seaofbtc.com/api/indicator/macd"
        data = urllib.parse.urlencode(values)
        data = data.encode('utf-8')
        req = urllib.request.Request(url, data)
        resp = urllib.request.urlopen(req)
        respData = resp.read()
        newData = str(respData).replace("b","").replace('[','').replace(']'

        #print(newData)

        split = newData.split('::')

        macd = split[0]
        macd = macd.split(", ")
        ema9 = ema9.split(", ")
        hist = hist.split(", ")


        try:
            macd = [float(i) for i in macd]
        except Exception as e:
            print(str(e)+"  macd")
        try:
            ema9 = [float(i) for i in ema9]
        except Exception as e:
            print(str(e)+"  ema9")
        try:
            hist = [float(i) for i in hist]
        except Exception as e:
            print(str(e)+"  hist")





        print("call!!!")




        if location == "top":
            try:
                a0.plot(OHLC['MPLDates'][fast:], macd[fast:], color=darkCol
                a0.plot(OHLC['MPLDates'][fast:], ema9[fast:], color=lightCo
                a0.fill_between(OHLC['MPLDates'][fast:], hist[fast:], 0, al
                datLabel = "MACD"
                a0.set_ylabel(datLabel)
            except Exception as e:
                print(str(e))
                topIndicator = "none"


        elif location == "bottom":
            try:
```

```
                a3.fill_between(OHLC['MPLDates'][fast:], hist[fast:], 0, al
                datLabel = "MACD"
                a3.set_ylabel(datLabel)
            except Exception as e:
                print(str(e))
                bottomIndicator = "none"

####################################
####################################

    def rsiIndicator(priceData,location="top"):

        if location == "top":
            values = {'key': 1,'prices':priceData,'periods':topIndicator[1]

        elif location == "bottom":
            values = {'key': 1,'prices':priceData,'periods':bottomIndicator


        url = "http://seaofbtc.com/api/indicator/rsi"
        data = urllib.parse.urlencode(values)
        data = data.encode('utf-8')
        req = urllib.request.Request(url, data)
        resp = urllib.request.urlopen(req)
        respData = resp.read()
        newData = str(respData).replace("b","").replace('[','').replace(']'
        priceList = newData.split(', ')
        rsiData = [float(i) for i in priceList]

        print("call!!!")


        if location == "top":
            a0.plot_date(OHLC['MPLDates'], rsiData,lightColor, label ="RSI"
            datLabel = "RSI("+str(topIndicator[1])+")"
            a0.set_ylabel(datLabel)

        elif location == "bottom":
            a3.plot_date(OHLC['MPLDates'], rsiData,lightColor, label ="RSI"
            datLabel = "RSI("+str(bottomIndicator[1])+")"
            a3.set_ylabel(datLabel)
```

```
def moving_average(x, n, type='simple'):

    x = np.asarray(x)
    if type=='simple':
        weights = np.ones(n)
    else:
        weights = np.exp(np.linspace(-1, 0, n))

    weights /= weights.sum()


    a =  np.convolve(x, weights, mode='full')[:len(x)]
    return a


if chartLoad:
    if paneCount == 1:
        if DataPace == "tick":
            try:
                if exchange == "BTC-e":
                    a = plt.subplot2grid((6,4), (0,0), rowspan=5, colsp
                    a2 = plt.subplot2grid((6,4), (5,0), rowspan=1, cols

                    dataLink = 'https://btc-e.com/api/3/trades/btc_usd?
                    data = urllib.request.urlopen(dataLink)
                    data = data.readall().decode('utf-8')
                    data = json.loads(data)
                    data = data["btc_usd"]
                    data = pd.DataFrame(data)


                    data["datestamp"] = np.array(data['timestamp']).ast
                    allDates = data["datestamp"].tolist()

                    buys = data[(data['type']=='bid')]
                    buyDates = (buys["datestamp"]).tolist()


                    sells = data[(data['type']=='ask')]
                    sellDates = (sells["datestamp"]).tolist()
                    a.clear()

                    a.plot_date(buyDates,buys["price"], lightColor, lab

                    a.plot_date(sellDates,sells["price"], darkColor, la

                    a2.fill_between(allDates,0, volume, facecolor='#183

                    a.xaxis.set_major_locator(mticker.MaxNLocator(5))
                    a.xaxis.set_major_formatter(mdates.DateFormatter('%
                    plt.setp(a.get_xticklabels(), visible=False)




                    a.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3
                             ncol=2, borderaxespad=0.)

                    title = 'Last Price: '+str(data["price"][1999])
                    a.set_title(title)

                if exchange == 'Bitstamp':
                    a = plt.subplot2grid((6,4), (0,0), rowspan=5, colsp
                    a2 = plt.subplot2grid((6,4), (5,0), rowspan=1, cols

                    dataLink = 'https://www.bitstamp.net/api/transactio
                    data = urllib.request.urlopen(dataLink)
                    data = data.readall().decode('utf-8')
                    data = json.loads(data)
                    data = pd.DataFrame(data)
                    data["datestamp"] = np.array(data['date'].apply(int
                    datestamps = data["datestamp"].tolist()
                    volume = data["amount"].apply(float).tolist()

                    a.clear()

                    a.plot_date(datestamps,data["price"], '#183A54')


                    a2.fill_between(datestamps,0, volume, facecolor='#1
                    plt.setp(a.get_xticklabels(), visible=False)

                    title = exchange+' Tick Data\nLast Price: '+str(dat
                    a.set_title(title)
                    priceData = data["price"].apply(float).tolist()


                if exchange == 'Bitfinex':
                    a = plt.subplot2grid((6,4), (0,0), rowspan=5, colsp
                    a2 = plt.subplot2grid((6,4), (5,0), rowspan=1, cols

                    dataLink = 'https://api.bitfinex.com/v1/trades/btcu

                    data = urllib.request.urlopen(dataLink)
                    data = data.readall().decode('utf-8')
                    data = json.loads(data)
                    data = pd.DataFrame(data)

                    volume = data["amount"].apply(float).tolist()


                    data["datestamp"] = np.array(data['timestamp']).ast
                    allDates = data["datestamp"].tolist()

                    buys = data[(data['type']=='buy')]
                    buyDates = (buys["datestamp"]).tolist()

                    sells = data[(data['type']=='sell')]
                    sellDates = (sells["datestamp"]).tolist()

                    a.clear()


                    a.plot_date(buyDates,buys["price"], lightColor, lab
                    a.plot_date(sellDates,sells["price"], darkColor, la
                    a2.fill_between(allDates,0, volume, facecolor='#183


                    a.xaxis.set_major_locator(mticker.MaxNLocator(5))
                    a.xaxis.set_major_formatter(mdates.DateFormatter('%
                    plt.setp(a.get_xticklabels(), visible=False)
                    a.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3
                    title = exchange+' Tick Data\nLast Price: '+str(dat
                    a.set_title(title)
                    priceData = data["price"].apply(float).tolist()

                if exchange == 'Huobi':
                    try:
                        a = plt.subplot2grid((6,4), (0,0), rowspan=6, c

                        data = urllib.request.urlopen('http://seaofbtc.

                        data = str(data).replace('b','').replace("'",''
                        data = json.loads(data)



                        dateStamp = np.array(data[0]).astype('datetime6
                        dateStamp = dateStamp.tolist()
                        print('here')

                        df = pd.DataFrame({'Datetime':dateStamp})




                        df['Price'] = data[1]

                        df['Volume'] = data[2]
                        df['Symbol'] = "BTCUSD"
                        df['MPLDate'] = df['Datetime'].apply(lambda dat
                        df = df.set_index('Datetime')
                        lastPrice = df['Price'][-1]

                        a.plot_date(df['MPLDate'][-4500:],df['Price'][-

                        a.xaxis.set_major_locator(mticker.MaxNLocator(5
                        a.xaxis.set_major_formatter(mdates.DateFormatte


                        title = exchange+' Tick Data\nLast Price: '+str
                        a.set_title(title)
                        priceData = df['Price'].apply(float).tolist()
                    except Exception as e:
                except Exception as e:
                    print("failed",str(e))
                    DatCounter = 9000


##### ALL OTHER, NON-TICK, DATA. ##################################
            else:
                if DatCounter > 12:
                    try:
                        if exchange == 'Huobi':
                            if topIndicator != "none":

                                a = plt.subplot2grid((6,4), (1,0), rowspan=
                                a0 = plt.subplot2grid((6,4), (0,0), sharex=
                            else:
                                a = plt.subplot2grid((6,4), (0,0), rowspan=

                        else:
                            if topIndicator != "none" and bottomIndicator !
                                # actual price chart.
                                a = plt.subplot2grid((6,4), (1,0), rowspan=
                                # volume!
                                a2 = plt.subplot2grid((6,4), (4,0), sharex=
                                # top indicator
                                a0 = plt.subplot2grid((6,4), (0,0), sharex=
                                # bottom indicator
                                a3 = plt.subplot2grid((6,4), (5,0), sharex=

                            elif topIndicator != "none":
                                a = plt.subplot2grid((6,4), (1,0), rowspan=
                                a2 = plt.subplot2grid((6,4), (5,0), sharex=
                                a0 = plt.subplot2grid((6,4), (0,0), sharex=
                            elif bottomIndicator != "none":
                                a = plt.subplot2grid((6,4), (0,0), rowspan=
                                a2 = plt.subplot2grid((6,4), (4,0), sharex=
                                #a0 = plt.subplot2grid((6,4), (0,0), sharex
                                a3 = plt.subplot2grid((6,4), (5,0), sharex=

                            else:
```

```
print('http://seaofbtc.com/api/basic/price?key=1&tf
data = urllib.request.urlopen('http://seaofbtc.com/





data = str(data).replace('b','').replace("'",'')
data = json.loads(data)

dateStamp = np.array(data[0]).astype('datetime64[s]
dateStamp = dateStamp.tolist()

df = pd.DataFrame({'Datetime':dateStamp})
df['Price'] = data[1]
df['Volume'] = data[2]
df['Symbol'] = "BTCUSD"
df['MPLDate'] = df['Datetime'].apply(lambda date: m
df = df.set_index('Datetime')


OHLC =  df['Price'].resample(resampleSize, how='ohl
OHLC = OHLC.dropna()

volumeData = df['Volume'].resample(resampleSize, ho

OHLC['dateCopy'] = OHLC.index
OHLC['MPLDates'] = OHLC['dateCopy'].apply(lambda da
del OHLC['dateCopy']

volumeData['dateCopy'] = volumeData.index
volumeData['MPLDates'] = volumeData['dateCopy'].app
del volumeData['dateCopy']


priceData = OHLC['close'].apply(float).tolist()

a.clear()
if middleIndicator != "none":
```

```
            if eachMA[0] == "sma":
                sma = pd.rolling_mean(OHLC["close"],eac
                label = str(eachMA[1])+" SMA"
                a.plot(OHLC['MPLDates'],sma, label=labe
            if eachMA[0] == "ema":
                ewma = pd.stats.moments.ewma
                label = str(eachMA[1])+" EMA"
                a.plot(OHLC['MPLDates'],ewma(OHLC["clos


        a.legend(loc=0)






    if topIndicator[0] == "rsi":
        rsiIndicator(priceData,"top")
    elif topIndicator == "macd":
        try:
            computeMACD(priceData,location="top")
        except:
            print("failed macd")




    if bottomIndicator[0] == "rsi":
        rsiIndicator(priceData,"bottom")
    elif bottomIndicator == "macd":
        try:
            computeMACD(priceData,location="bottom")
        except:
            print("failed macd")
```

```
if exchange != 'Huobi':
    a2.fill_between(volumeData['MPLDates'],0, volum
    a2.set_ylabel("volume")


a.xaxis.set_major_locator(mticker.MaxNLocator(3))
a.xaxis.set_major_formatter(mdates.DateFormatter('%

plt.setp(a.get_xticklabels(), visible=False)

if topIndicator != "none":
    plt.setp(a0.get_xticklabels(), visible=False)

if bottomIndicator != "none":
    plt.setp(a2.get_xticklabels(), visible=False)

x = (len(OHLC['close']))-1

if DataPace == '1d':
    title = exchange+' 1 Day Data with '+resampleSi
if DataPace == '3d':
    title = exchange+' 3 Day Data with '+resampleSi
if DataPace == '7d':
    title = exchange+' 7 Day Data with '+resampleSi


if topIndicator != "none":
    a0.set_title(title)
else:
    a.set_title(title)
print('NewGraph!')

DatCounter = 0
```

```
                    DatCounter = 9000
            else:
                DatCounter += 1




class SeaofBTCapp(tk.Tk):

    def __init__(self, *args, **kwargs):

        tk.Tk.__init__(self, *args, **kwargs)
        tk.Tk.wm_title(self, "Sea of BTC Client")

        container = tk.Frame(self)
        container.pack(side="top", fill="both", expand = True)
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)



        menubar = tk.Menu(container)
        filemenu = tk.Menu(menubar, tearoff=0)
        filemenu.add_command(label="Save settings", command=lambda: popupms
        filemenu.add_separator()
        filemenu.add_command(label="Exit", command=quit)
        menubar.add_cascade(label="File", menu=filemenu)


        exchangeChoice = tk.Menu(menubar, tearoff=1)
        exchangeChoice.add_command ( label="BTC-e",
                                    command=lambda: changeExchange('BTC-e','b
        exchangeChoice.add_command ( label="Bitfinex",
                                    command=lambda: changeExchange('Bitfinex'
        exchangeChoice.add_command ( label="Bitstamp",
                                    command=lambda: changeExchange('Bitstamp'
        exchangeChoice.add_command ( label="Huobi",
        dataTF = tk.Menu(menubar, tearoff=1)
        dataTF.add_command ( label="Tick",
                                    command=lambda: changeTimeFrame('tick') )
        dataTF.add_command ( label="1 day",
                                    command=lambda: changeTimeFrame('1d') )
        dataTF.add_command ( label="3 day",
                                    command=lambda: changeTimeFrame('3d') )
        dataTF.add_command ( label="1 Week",
                                    command=lambda: changeTimeFrame('7d') )
        menubar.add_cascade(label = "Data Time Frame", menu = dataTF)


        OHLCI = tk.Menu(menubar, tearoff=1)

        OHLCI.add_command ( label="Tick",
                                    command=lambda: changeTimeFrame('tick') )
        OHLCI.add_command ( label="1 minute",
                                    command=lambda: changeSampleSize('1Min',0
        OHLCI.add_command ( label="5 minute",
                                    command=lambda: changeSampleSize('5Min',0
        OHLCI.add_command ( label="15 minute",
                                    command=lambda: changeSampleSize('15Min',
        OHLCI.add_command ( label="30 minute",
                                    command=lambda: changeSampleSize('30Min',
        OHLCI.add_command ( label="1 Hour",
                                    command=lambda: changeSampleSize('1H',0.0
        OHLCI.add_command ( label="3 Hour",
                                    command=lambda: changeSampleSize('3H',0.0
        menubar.add_cascade(label = "OHLC Interval", menu = OHLCI)


        topIndi = tk.Menu(menubar, tearoff=1)
        topIndi.add_command(label="None",
                                    command=lambda: addTopIndicator('none'))
        topIndi.add_separator()
        topIndi.add_command ( label="RSI",
                                    command=lambda: addTopIndicator('rsi'))
        topIndi.add_command ( label="MACD",
                                    command=lambda: addTopIndicator('macd'))
```

```
mainI = tk.Menu(menubar, tearoff=1)
mainI.add_command ( label="None",
                          command=lambda: addMiddleIndicator('none'
mainI.add_separator()
mainI.add_command ( label="SMA",
                          command=lambda: addMiddleIndicator('sma')
mainI.add_command ( label="EMA",
                          command=lambda: addMiddleIndicator('ema')
menubar.add_cascade(label = "Main Graph Indicator", menu = mainI)



bottomI = tk.Menu(menubar, tearoff=1)
bottomI.add_command ( label="None",
                          command=lambda: addBottomIndicator('none'
bottomI.add_separator()
bottomI.add_command ( label="RSI",
                          command=lambda: addBottomIndicator('rsi')
bottomI.add_command ( label="MACD",
                          command=lambda: addBottomIndicator('macd'
menubar.add_cascade(label = "Bottom Indicator", menu = bottomI)


tradeButton = tk.Menu(menubar, tearoff=1)
tradeButton.add_command ( label="Manual Trading",
                          command=lambda: print('NOT live yet'))
tradeButton.add_separator()
tradeButton.add_command ( label="Automated Trading",
                          command=lambda: print('NOT live yet'))

tradeButton.add_separator()
tradeButton.add_command ( label="Quick Buy",
                          command=lambda: print('quick buy!'))#, ac
tradeButton.add_command ( label="Quick Sell",
                          command=lambda: print('quick sell'))#, ac
tradeButton.add_separator()
tradeButton.add_command ( label="Set-up Quick Buy/Sell",
                          command=lambda: print('quick buy!'))
```

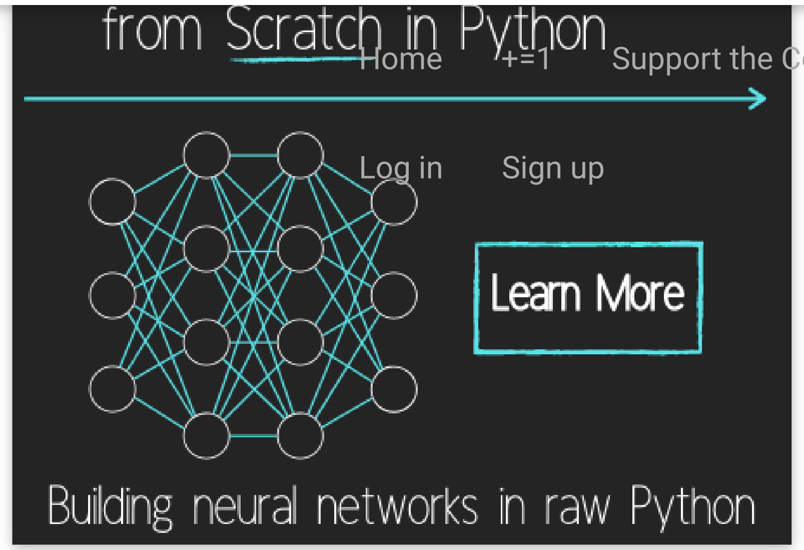
```
        startStop = tk.Menu(menubar, tearoff=1)
        startStop.add_command ( label="Resume",
                                command=lambda: loadChart('start'))
        startStop.add_command ( label="Pause",
                                command=lambda: loadChart('stop'))
        menubar.add_cascade(label = "Resume/Pause Client", menu = startStop



        helpmenu = tk.Menu(menubar, tearoff=0)
        helpmenu.add_command(label="Tutorial", command=tutorial)
        menubar.add_cascade(label="Help", menu=helpmenu)

        tk.Tk.config(self, menu=menubar)


        self.frames = {}
        for F in (StartPage, BTCe_Page):

            frame = F(container, self)
            self.frames[F] = frame
            frame.grid(row=0, column=0, sticky="nsew")

        self.show_frame(StartPage)

        tk.Tk.iconbitmap(self,default='clienticon.ico')

    def show_frame(self, cont):

        frame = self.frames[cont]
        frame.tkraise()


class StartPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self,parent)
        your API keys into the program. We, as in Sea of BTC, never
        see your API information. The program may save them locally, howeve
        to make things easier on you. Keep in mind that it is a fantastic i
        to enable 'IP Whitelisting' if your exchange supports it, and only
        trading via your specific IP address. On most exchanges, even if so
        was to acquire your API key, withdrawals are not possible. Some sti
        give the option, so make sure this is turned OFF if your exchange a

        Sea of BTC makes no promise of warranty, satisfaction, performance,
        anything else. Understand that your use of this client is completel
        at your own risk.""", font=LARGE_FONT)




        label.pack(side="top", fill="x", pady=10)

        button1 = ttk.Button(self, text="Agree",
                            command=lambda: controller.show_frame(BTCe_Page
        button2 = ttk.Button(self, text="Disagree",
                            command=quit)
        button1.pack()
        button2.pack()

class BTCe_Page(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label = ttk.Label(self, text="BTC-e Exchange Page", font=LARGE_FONT
        label.pack(pady=10,padx=10)


        canvas = FigureCanvasTkAgg(f, self)
        canvas.show()
        canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)

        toolbar = NavigationToolbar2TkAgg( canvas, self )
        toolbar.update()
        canvas._tkcanvas.pack(side=tk.TOP, fill=tk.BOTH, expand=1)
```

```
ani = animation.FuncAnimation(f,animate, interval=5000)
app.mainloop()
```

There exists **1 challenge(s)** for this tutorial. Sign Up To **+=1** for access to these, video downloads, and no ads.

There exists **2 quiz/question(s)** for this tutorial. Sign Up To **+=1** for access to these, video downloads, and no ads.

That's it for this series. For more tutorials, head to the Home Page


from Scratch in Python
Learn More
Building neural networks in raw Python

You've reached the end!