



search

Home

+=1

Support the Content

Community

Log in

Sign up

Setting up sub plots within our Tkinter GUI

Setting up sub plots - Tkinter tutorial Python 3.4 part 25



The code for changing pages was derived from: <http://stackoverflow.com/questions/7546050/switch-between-two->
License: <http://creativecommons.org/licenses/by-sa/3.0/>

```
import matplotlib
matplotlib.use("TkAgg")
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2TkAgg
#from matplotlib.figure import Figure
import matplotlib.animation as animation
from matplotlib import style
from matplotlib import pyplot as plt
```

```
import matplotlib.dates as mdates
import matplotlib.ticker as mticker
from matplotlib.finance import candlestick_ohlc
```

```
import tkinter as tk
from tkinter import ttk
```

```
import urllib
import json
```

```
import pandas as pd
import numpy as np
```

```
LARGE_FONT= ("Verdana", 12)
```



```

style.use("ggplot")

f = plt.figure()

#a = f.add_subplot(111)

exchange = "BTC-e"
DatCounter = 9000
programName = "btce"
resampleSize = "15Min"
DataPace = "tick"
candleWidth = 0.008

paneCount = 1

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

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

EMAs = []
SMAs = []

def tutorial():

    ## def Leavemini(what):
    ##     what.destroy()

    def page2():
        tut.destroy()
        tut2 = tk.Tk()

        def page3():
            tut2.destroy()
            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()
            tut2.mainloop()

        tut = tk.Tk()
        tut.wm_title("Tutorial")

```



```
B1 = ttk.Button(tut, text = "Overview of the application", command=page2)
B1.pack()

B2 = ttk.Button(tut, text = "How do I trade with this client?", command=lambda:popupmsg("Not yet completed"))
B2.pack()

B3 = ttk.Button(tut, text = "Indicator Questions/Help", command=lambda:popupmsg("Not yet completed"))
B3.pack()

tut.mainloop()
```

```
def loadChart(run):
    global chartLoad
```

```
    if run == "start":
        chartLoad = True
```

```
    elif run == "stop":
        chartLoad = False
```

```
def addMiddleIndicator(what):
    global middleIndicator
    global DatCounter
```

```
    if DataPace == "tick":
        popupmsg("Indicators in Tick Data not available.")
```

```
    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 want your SMA to be.")
                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("sma")
            group.append(int(periods))
```



```

print("middle indicator set to:",middleIndicator)
midIQ.destroy()

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

if what == "ema":
    midIQ = tk.Tk()
    #midIQ.wm_title("Periods?")
    label = ttk.Label(midIQ, text="Choose how many periods you want your EMA to be.")
    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("middle indicator set to:",middleIndicator)
        midIQ.destroy()

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

else:
    if what == "sma":
        midIQ = tk.Tk()
        midIQ.wm_title("Periods?")
        label = ttk.Label(midIQ, text="Choose how many periods you want your SMA to be.")
        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("sma")
            group.append(int(periods))
            middleIndicator.append(group)
            DatCounter = 9000
            print("middle indicator set to:",middleIndicator)
            midIQ.destroy()

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

```



```

    if what == "ema":
        midIQ = tk.Tk()
        midIQ.wm_title("Periods?")
        label = ttk.Label(midIQ, text="Choose how many periods you want your EMA to be.")
        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("middle indicator set to:",middleIndicator)
            midIQ.destroy()

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

    else:
        middleIndicator = "none"

def addTopIndicator(what):
    global topIndicator
    global DatCounter

    if DataPace == "tick":
        popupmsg("Indicators in Tick Data not available.")

    elif 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 RSI calculation to consider.")
        label.pack(side="top", fill="x", pady=10)

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

        def callback():
            global topIndicator
            global DatCounter

```



```

        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 addBottomIndicator(what):
    global bottomIndicator
    global DatCounter

    if DataPace == "tick":
        popupmsg("Indicators in Tick Data not available.")

    elif 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 RSI calculation to consider.")
        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 bottom indicator to",group)
        rsiQ.destroy()

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

elif what == "macd":
    global bottomIndicator

```

**DatCounter** = 9000

```

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

def changeSampleSize(size,width):
    global resampleSize
    global DatCounter
    global candleWidth
    if DataPace == "7d" and resampleSize == "1Min":
        popupmsg("Too much data chosen, choose a smaller time frame or higher OHLC interval")

    elif DataPace == "tick":
        popupmsg("You're currently viewing tick data, not OHLC.")

    else:
        resampleSize = size
        DatCounter = 9000
        candleWidth = width

def changeExchange(towhat,pn):
    global exchange
    global DatCounter
    global programName

    exchange = towhat
    programName = pn
    DatCounter = 9000

def popupmsg(msg):
    popup = tk.Tk()
    popup.wm_title("!")
    label = ttk.Label(popup, text=msg, font=NORM_FONT)
    label.pack(side="top", fill="x", pady=10)
    B1 = ttk.Button(popup, text="Okay", command = popup.destroy)
    B1.pack()
    popup.mainloop()

def animate(i):
    global refreshRate
    global DatCounter

    if chartLoad:
        if paneCount == 1:

            if DataPace == "tick":
                try:
                    if exchange == "BTC-e":

```



```

dataLink = 'https://btc-e.com/api/3/trades/btc_usd?limit=2000'
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']).astype("datetime64[s]")
allDates = data["datestamp"].tolist()

buys = data[(data['type']=="bid")]
#buys["datestamp"] = np.array(buys["timestamp"]).astype("datetime64[s]")
buyDates = (buys["datestamp"]).tolist()

sells = data[(data['type']=="ask")]
#sells["datestamp"] = np.array(sells["timestamp"]).astype("datetime64[s]")
sellDates = (sells["datestamp"]).tolist()

volume = data["amount"]

a.clear()

a.plot_date(buyDates, buys["price"], lightColor, label="buys")
a.plot_date(sellDates, sells["price"], darkColor, label="sells")

a2.fill_between(allDates, 0, volume, facecolor = darkColor)

a.xaxis.set_major_locator(mticker.MaxNLocator(5))
a.xaxis.set_major_formatter(mdates.DateFormatter("%Y-%m-%d %H:%M:%S"))
plt.setp(a.get_xticklabels(), visible = False)

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

title = "BTC-e BTCUSD Prices\nLast Price: "+str(data["price"][1999])
a.set_title(title)
priceData = data['price'].apply(float).tolist()

if exchange == "Bitstamp":
    a = plt.subplot2grid((6,4), (0,0), rowspan = 5, colspan = 4)
    a2 = plt.subplot2grid((6,4), (5,0), rowspan = 1, colspan = 4, sharex = a)

    dataLink = 'https://www.bitstamp.net/api/transactions/'
    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)).astype("datetime64[s]")
    dateStamps = data["datestamp"].tolist()
    #allDates = data["datestamp"].tolist()

    buys = data[(data['type']=="bid")]
    #buys["datestamp"] = np.array(buys["timestamp"]).astype("datetime64[s]")
    buyDates = (buys["datestamp"]).tolist()

```

```

##
##
##
##

```




```

##      #sells["datestamp"] = np.array(sells["timestamp"]).astype("datetime64[s]")
##      sellDates = (sells["datestamp"]).tolist()

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

a.clear()

a.plot_date(dateStamps, data["price"], lightColor, label="buys")

a2.fill_between(dateStamps, 0, volume, facecolor = darkColor)

a.xaxis.set_major_locator(mticker.MaxNLocator(5))
a.xaxis.set_major_formatter(mdates.DateFormatter("%Y-%m-%d %H:%M:%S"))
plt.setp(a.get_xticklabels(), visible = False)

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

title = "Bitstamp BTCUSD Prices\nLast Price: "+str(data["price"][0])
a.set_title(title)
priceData = data['price'].apply(float).tolist()

if exchange == "Bitfinex":
    a = plt.subplot2grid((6,4), (0,0), rowspan = 5, colspan = 4)
    a2 = plt.subplot2grid((6,4), (5,0), rowspan = 1, colspan = 4, sharex = a)

    dataLink = 'https://api.bitfinex.com/v1/trades/btcusd?limit=2000'
    data = urllib.request.urlopen(dataLink)
    data = data.readall().decode("utf-8")
    data = json.loads(data)

    data = pd.DataFrame(data)

    data["datestamp"] = np.array(data['timestamp']).astype("datetime64[s]")
    allDates = data["datestamp"].tolist()

    buys = data[(data['type']=="buy")]
    #buys["datestamp"] = np.array(buys["timestamp"]).astype("datetime64[s]")
    buyDates = (buys["datestamp"]).tolist()

    sells = data[(data['type']=="sell")]
    #sells["datestamp"] = np.array(sells["timestamp"]).astype("datetime64[s]")
    sellDates = (sells["datestamp"]).tolist()

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

    a.clear()

    a.plot_date(buyDates, buys["price"], lightColor, label="buys")
    a.plot_date(sellDates, sells["price"], darkColor, label="sells")

    a2.fill_between(allDates, 0, volume, facecolor = darkColor)

    a.xaxis.set_major_locator(mticker.MaxNLocator(5))
    a.xaxis.set_major_formatter(mdates.DateFormatter("%Y-%m-%d %H:%M:%S"))
    plt.setp(a.get_xticklabels(), visible = False)

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

```



```

title = "Bitcoin BTCUSD Prices\nLast Price: "+str(data["price"][0])
a.set_title(title)
priceData = data['price'].apply(float).tolist()

if exchange == "Huobi":
    a = plt.subplot2grid((6,4), (0,0), rowspan = 6, colspan = 4)
    data = urllib.request.urlopen('http://seaofbtc.com/api/basic/price?key=1&tf=1d&exchange=')
    data = data.decode()

    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: mdates.date2num(date.to_pydatetime()))

    df = df.set_index("Datetime")

    lastPrice = df["Price"][-1]

    a.plot_date(df['MPLDate'][-4500:], df['Price'][-4500:], lightColor, label="price")

    a.xaxis.set_major_locator(mticker.MaxNLocator(5))
    a.xaxis.set_major_formatter(mdates.DateFormatter("%Y-%m-%d %H:%M:%S"))

    title = "Huobi BTCUSD Prices\nLast Price: "+str(lastPrice)
    a.set_title(title)

    priceData = df['price'].apply(float).tolist()

except Exception as e:
    print("Failed because of:",e)

else:
    if DatCounter > 12:
        try:
            if exchange == "Huobi":
                if topIndicator != "none":
                    a = plt.subplot2grid((6,4),(1,0), rowspan=5, colspan = 4)
                    a2 = plt.subplot2grid((6,4),(0,0),sharex=a, rowspan=1, colspan = 4)
                else:
                    a = plt.subplot2grid((6,4),(0,0), rowspan=6, colspan = 4)

            else:
                if topIndicator != "none" and bottomIndicator != "none":
                    # Main Graph
                    a = plt.subplot2grid((6,4), (1,0), rowspan = 3, colspan = 4)

                    # Volume
                    a2 = plt.subplot2grid((6,4), (4,0), sharex = a, rowspan = 1, colspan = 4)

                    # Bottom Indicator
                    a3 = plt.subplot2grid((6,4), (5,0), sharex = a, rowspan = 1, colspan = 4)

                    # Top Indicator
                    a0 = plt.subplot2grid((6,4), (0,0), sharex = a, rowspan = 1, colspan = 4)

```



```

a = plt.subplot2grid((6,4), (1,0), rowspan = 4, colspan = 4)

# Volume
a2 = plt.subplot2grid((6,4), (5,0), sharex = a, rowspan = 1, colspan = 4)

# Top Indicator
a0 = plt.subplot2grid((6,4), (0,0), sharex = a, rowspan = 1, colspan = 4)

elif bottomIndicator != "none":

    # Main Graph
    a = plt.subplot2grid((6,4), (0,0), rowspan = 4, colspan = 4)

    # Volume
    a2 = plt.subplot2grid((6,4), (4,0), sharex = a, rowspan = 1, colspan = 4)

    # Bottom Indicator
    a3 = plt.subplot2grid((6,4), (5,0), sharex = a, rowspan = 1, colspan = 4)

else:

    # Main Graph
    a = plt.subplot2grid((6,4), (0,0), rowspan = 5, colspan = 4)

    # Volume
    a2 = plt.subplot2grid((6,4), (5,0), sharex = a, rowspan = 1, colspan = 4)

data = urllib.request.urlopen("http://seafbtc.com/api/basic/price?key=1&tf="+DataPac
data = data.decode()
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: mdates.date2num(date.to_pydatetime())
df = df.set_index("Datetime")

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

volumeData = df['Volume'].resample(resampleSize, how={'volume':'sum'})

OHLC["dateCopy"] = OHLC.index
OHLC["MPLDates"] = OHLC["dateCopy"].apply(lambda date: mdates.date2num(date.to_pydate

del OHLC["dateCopy"]

volumeData["dateCopy"] = volumeData.index
volumeData["MPLDates"] = volumeData["dateCopy"].apply(lambda date: mdates.date2num(dat

del volumeData["dateCopy"]

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

a.clear()

if middleIndicator != "none":

```



```
if eachMA[0] == "sma":
    sma = pd.rolling_mean(OHLC["close"], eachMA[1])
    label = str(eachMA[1]+" SMA")
    a.plot(OHLC["MPLDates"], sma, label=label)

if eachMA[0] == "ema":
    ewma = pd.stats.moments.ewma
    label = str(eachMA[1]+" EMA")
    a.plot(OHLC["MPLDates"], ewma(OHLC["close"], eachMA[1]), label=label)

a.legend(loc=0)

if topIndicator[0] == "rsi":
    rsiIndicator(priceData,"top")

elif topIndicator == "macd":
    try:
        computeMACD(priceData, location = "top")

    except Exception as e:
        print(str(e))

if bottomIndicator[0] == "rsi":
    rsiIndicator(priceData,"bottom")

elif bottomIndicator == "macd":
    try:
        computeMACD(priceData, location = "bottom")

    except Exception as e:
        print(str(e))
```



```
except Exception as e:
    print('failed in the non-tick animate:',str(e))
```

```
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: popupmsg("Not supported just yet!"))
```

```
        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","btce"))
```

```
        exchangeChoice.add_command(label="Bitfinex",
```

```
                                   command=lambda: changeExchange("Bitfinex","bitfinex"))
```

```
        exchangeChoice.add_command(label="Bitstamp",
```

```
                                   command=lambda: changeExchange("Bitstamp","bitstamp"))
```

```
        exchangeChoice.add_command(label="Huobi",
```

```
                                   command=lambda: changeExchange("Huobi","huobi"))
```

```
        menubar.add_cascade(label="Exchange", menu=exchangeChoice)
```

```
        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', 0.0005))
```



```

OHLCI.add_command(label = "15 minute",
                   command=lambda: changeSampleSize('15Min', 0.008))
OHLCI.add_command(label = "30 minute",
                   command=lambda: changeSampleSize('30Min', 0.016))
OHLCI.add_command(label = "1 Hour",
                   command=lambda: changeSampleSize('1H', 0.032))
OHLCI.add_command(label = "3 Hour",
                   command=lambda: changeSampleSize('3H', 0.096))

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_command(label="RSI",
                    command = lambda: addTopIndicator('rsi'))
topIndi.add_command(label="MACD",
                    command = lambda: addTopIndicator('macd'))

menubar.add_cascade(label="Top Indicator", menu=topIndi)

mainI = tk.Menu(menubar, tearoff=1)
mainI.add_command(label="None",
                  command = lambda: addMiddleIndicator('none'))
mainI.add_command(label="SMA",
                  command = lambda: addMiddleIndicator('sma'))
mainI.add_command(label="EMA",
                  command = lambda: addMiddleIndicator('ema'))

menubar.add_cascade(label="Main/middle Indicator", menu=mainI)

bottomI = tk.Menu(menubar, tearoff=1)
bottomI.add_command(label="None",
                    command = lambda: addBottomIndicator('none'))
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: popupmsg("This is not live yet"))
tradeButton.add_command(label = "Automated Trading",
                        command=lambda: popupmsg("This is not live yet"))

tradeButton.add_separator()
tradeButton.add_command(label = "Quick Buy",
                        command=lambda: popupmsg("This is not live yet"))
tradeButton.add_command(label = "Quick Sell",
                        command=lambda: popupmsg("This is not live yet"))

tradeButton.add_separator()
tradeButton.add_command(label = "Set-up Quick Buy/Sell",
                        command=lambda: popupmsg("This is not live yet"))

```



```
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)
        label = tk.Label(self, text=("""ALPHA Bitcoin trading application
use at your own risk. There is no promise
of warranty."""), font=LARGE_FONT)
        label.pack(pady=10, padx=10)

        button1 = ttk.Button(self, text="Agree",
                             command=lambda: controller.show_frame(BTCE_Page))
        button1.pack()

        button2 = ttk.Button(self, text="Disagree",
                             command=quit)
        button2.pack()
```



```

def __init__(self, parent, controller):
    tk.Frame.__init__(self, parent)
    label = tk.Label(self, text="Page One!!!", font=LARGE_FONT)
    label.pack(pady=10, padx=10)

    button1 = ttk.Button(self, text="Back to Home",
                        command=lambda: controller.show_frame(StartPage))
    button1.pack()

class BTCe_Page(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label = tk.Label(self, text="Graph Page!", font=LARGE_FONT)
        label.pack(pady=10, padx=10)

        button1 = ttk.Button(self, text="Back to Home",
                            command=lambda: controller.show_frame(StartPage))
        button1.pack()

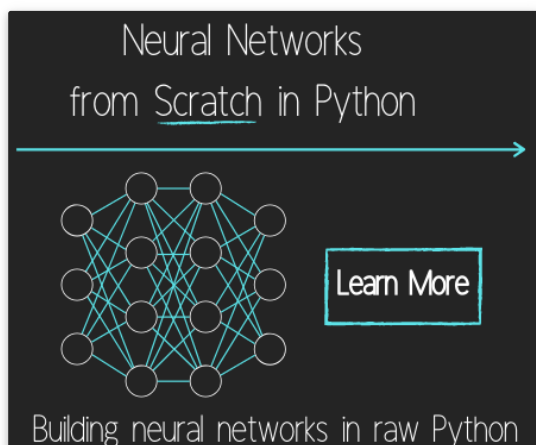
        canvas = FigureCanvasTkAgg(f, self)
        canvas.show()
        canvas.get_tk_widget().pack(side=tk.BOTTOM, fill=tk.BOTH, expand=True)

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

app = SeaofBTCapp()
app.geometry("1280x720")
ani = animation.FuncAnimation(f, animate, interval=5000)
app.mainloop()

```

The next tutorial: [Graphing An OHLC Candlestick Graph Embedded In Our Tkinter GUI](#)





Object Oriented Programming Crash Course with Tkinter

Passing functions with Parameters in Tkinter using Lambda

How to change and show a new window in Tkinter

Styling your GUI a bit using TTK

How to embed a Matplotlib graph to your Tkinter GUI

How to make the Matplotlib graph live in your application

Organizing our GUI

Plotting Live Updating Data in Matplotlib and our Tkinter GUI

Customizing an embedded Matplotlib Graph in Tkinter

Creating our Main Menu in Tkinter

Building a pop-up message window

Exchange Choice Option

Time-frame and sample size option

Adding indicator Menus (3 videos)

Trading option, start/stop, and help menu options

Tutorial on adding a tutorial

Allowing the exchange choice option to affect actual shown exchange

Adding exchange choice cont'd

Adding exchange choices part 3

Indicator Support

Pulling data from the Sea of BTC API

Setting up sub plots within our Tkinter GUI

Graphing an OHLC candlestick graph embedded in our Tkinter GUI

Acquiring RSI data from Sea of BTC API

Acquiring MACD data from Sea of BTC API

Converting Tkinter application to .exe and installer with cx_Freeze



Visit The adidas® Store

adidas Flagship Store New York
[New York](#) 10AM–8PM

\$5.58

\$7.97

\$3.28

Shopping made easy on Temu
Temu

You've reached the end!

Contact: Harrison@pythonprogramming.net.

Support this Website!
Consulting and Contracting
[Facebook](#)
[Twitter](#)
[Instagram](#)

Legal stuff:
[Terms and Conditions](#)
[Privacy Policy](#)