Chapter 6:
Create A U.S. Stock Market Watch

In this chapter, you'll learn how to create a graphical live market watch for the U.S. stock market. If you run the program during the trading hours when the U.S. stock market is open, you'll see a graphical display of the major stock indexes and a couple of stocks that you are interested in.

To build up the necessary skills, you'll first learn to create a graphical Bitcoin watch using the *tkinter* module to display live price information. Whenever the price changes, the program will notify you. Further, if the price moves outside the pre-set upper or lower bounds, the program will alert you.

You can generalize the techniques to other financial markets such as the world stock market or the U.S. treasury bond market.

Bitcoin Watch

We start with Bitcoin because the Bitcoin price is updated 24/7, unlike the U.S. stock market where you can see live price updates only when the US stock market is open. In the process of creating a Bitcoin watch, you'll learn the necessary skills to building a market watch in other financial markets.

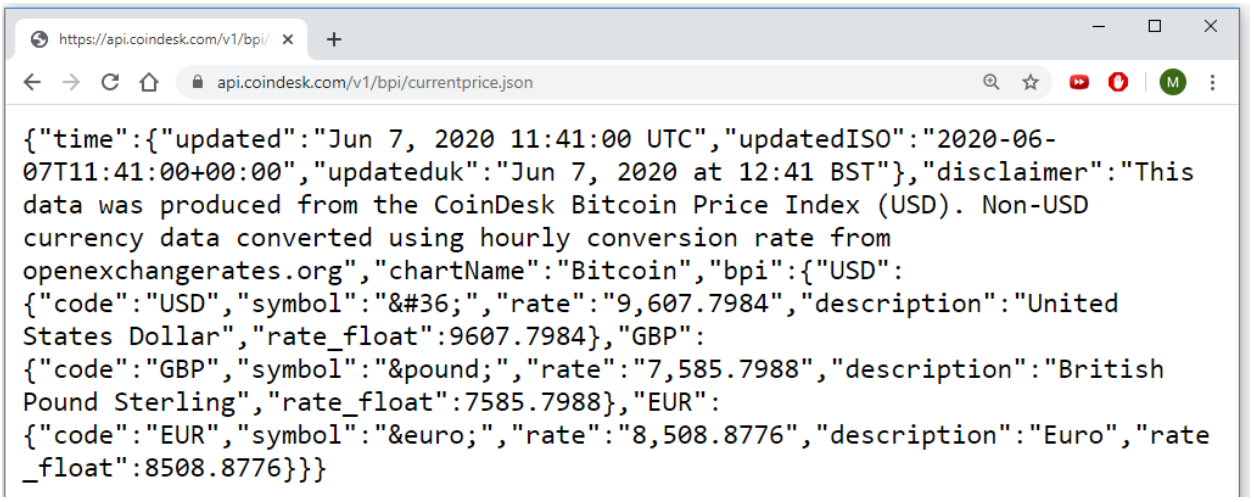
You'll first learn how to read JavaScript Object Notation (JSON) data and some basics of the *tkinter* module before you create a Bitcoin watch.

How to Read JSON Data

You can obtain the live Bitcoin price online for free. The price updates every minute or so, and it's updated 24/7.

To access bitcoin price using the Python program, you can use this url: *https://api.coindesk.com/v1/bpi/currentprice.json*. If you open the url with a web browser, you will see the price information similar to Figure 6-1.

INSERT FIGURE 6-1 HERE


https://api.coindesk.com/v1/bpi/
api.coindesk.com/v1/bpi/currentprice.json

```
{"time":{"updated":"Jun 7, 2020 11:41:00 UTC","updatedISO":"2020-06-
07T11:41:00+00:00","updateduk":"Jun 7, 2020 at 12:41 BST"},"disclaimer":"This
data was produced from the CoinDesk Bitcoin Price Index (USD). Non-USD
currency data converted using hourly conversion rate from
openexchangerates.org","chartName":"Bitcoin","bpi":{"USD":
{"code":"USD","symbol":"&#36;","rate":"9,607.7984","description":"United
States Dollar","rate_float":9607.7984},"GBP":
{"code":"GBP","symbol":"&pound;","rate":"7,585.7988","description":"British
Pound Sterling","rate_float":7585.7988},"EUR":
{"code":"EUR","symbol":"&euro;","rate":"8,508.8776","description":"Euro","rate
_float":8508.8776}}}
```

Figure 6-1: Live online information about Bitcoin price

The above data is hard to read and understand. You see dictionaries nested in a dictionary, which in turn is nested in another dictionary. It's hard to tell where one dictionary starts and ends.

The above data is in a JSON format. JSON is short for JavaScript Object Notation. It is a file format used for browser-server communication.

NOTE: For more information about JSON data format, see here https://en.wikipedia.org/wiki/JSON.

To make the JSON data easier to read, you should search for an online JSON data formatter to make it easier to understand. One good example is *https://jsonformatter.curiousconcept.com/*. If you open the url, you see a screen similar to Figure 6-2.

INSERT FIGURE 6-2 HERE

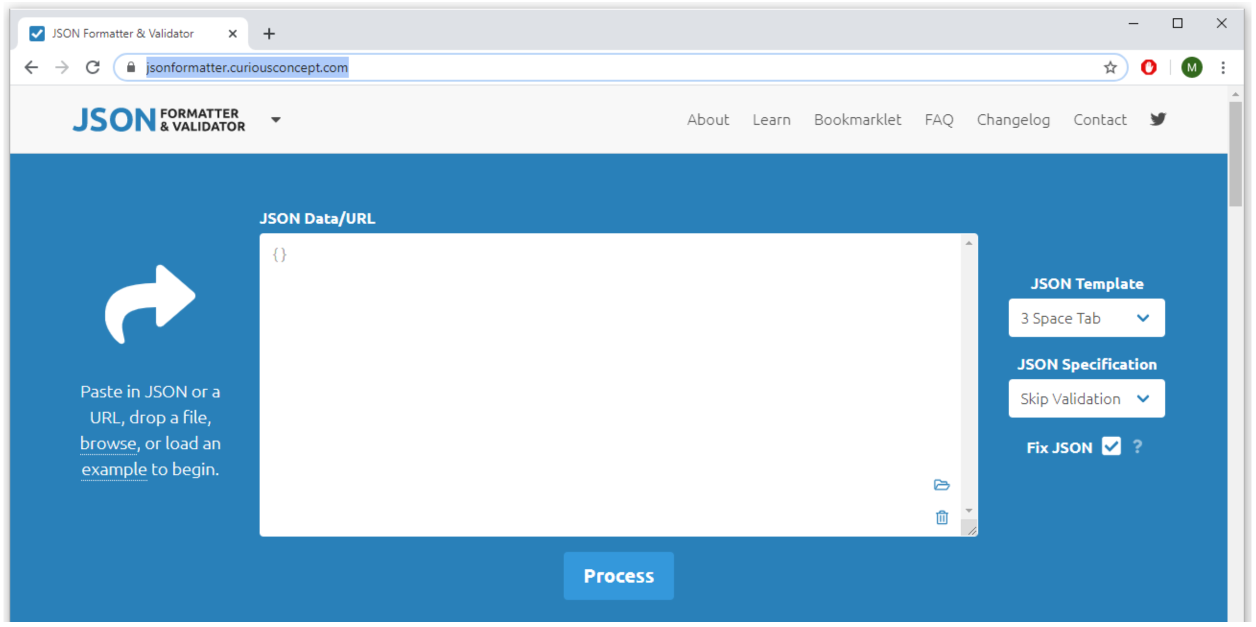

JSON Formatter & Validator
jsonformatter.curiousconcept.com
JSON FORMATTER & VALIDATOR
About
Learn
Bookmarklet
FAQ
Changelog
Contact
JSON Data/URL
{}
Paste in JSON or a URL, drop a file, browse, or load an example to begin.
JSON Template
3 Space Tab
JSON Specification
Skip Validation
Fix JSON
Process

Figure 6-2: A website to format JSON data

You can paste the JSON data in the designated space and press the `Process` button. It will convert the data into a much readable format as shown in Listing 6-1.

```
{
❶    "time":{
       "updated":"Jun 7, 2020 11:41:00 UTC",
       "updatedISO":"2020-06-07T11:41:00+00:00",
       "updateduk":"Jun 7, 2020 at 12:41 BST"

},
❷    "disclaimer":"This data was produced from the CoinDesk Bitcoin Price Inde
x (USD). Non-
USD currency data converted using hourly conversion rate from openexchangerat
es.org",
❸    "chartName":"Bitcoin",
❹    "bpi":{
❺       "USD":{
```

```
            "code":"USD",
            "symbol":"&#36;",
            "rate":"9,607.7984",
            "description":"United States Dollar",
            "rate_float":9607.7984

},
❻       "GBP":{
            "code":"GBP",
            "symbol":"&pound;",
            "rate":"7,585.7988",
            "description":"British Pound Sterling",
            "rate_float":7585.7988

},
❼       "EUR":{
            "code":"EUR",
            "symbol":"&euro;",
            "rate":"8,508.8776",
            "description":"Euro",
            "rate_float":8508.8776

}

}
}
```

Listing 6-1: The formatted JSON data about Bitcoin price

The dataset is a large dictionary with four elements, with key values `time` ❶, `disclaimer` ❷, `chartName` ❸, and `bpi` ❹, respectively. The value for the `bpi` key, in turn, is another dictionary with three keys: `USD` ❺, `GBP` ❻, and `EUR` ❼, respectively. They represent the Bitcoin price in U.S. dollars, British pounds, and Euros, respectively.

We want the Bitcoin price in U.S. dollars. The program *BitcoinPrice.py*, as shown in Listing 6-2, retrieves the bitcoin price and prints it out.

```
import requests
```

```
# specify the url to find the bitcoin price
url = 'https://api.coindesk.com/v1/bpi/currentprice.json'
#get the live information from bitcoin url
response = requests.get(url)   #❶
# read the JSON data
response_json = response.json()   #❷
# obtain the USD dictionary
usd=response_json['bpi']['USD']   #❸
# get the price
price=usd['rate_float']   #❹
print(f"The Bitcoin price is ${price}.")   #❺
```

Listing 6-2: The program to retrieve Bitcoin price

We first import the *requests* module and specify the url for the live Bitcoin price. We then use the `get()` method in the *requests* module to obtain the information from the website ❶. The `json()` method in the *requests* module reads the information into the JSON format ❷. We then extract the USD dictionary that contains all the Bitcoin price information in U.S. dollars ❸. The value we need from the dictionary is the price, and we use the key `rate_float` to retrieve it ❹.

Finally, we print out the bitcoin price ❺, and the output is as follows:

```
The Bitcoin price is $9607.7984.
```

BEGIN BOX

Try it Out

6-1. Run the program *BitcoinPrice.py* and do a Google search online about the Bitcoin price. Compare the two prices and see if they are close to each other.

END BOX

A Quick Introduction to the tkinter Module

tkinter is short for Tk interface. It's Python's default standard *graphical user interface* (GUI) package. Tk is a free and open-source software package that provides a cross-platform widget toolkit for building GUI in many programming languages, including Python.

NOTE: For more information about Tk and tkinter, visit https://en.wikipedia.org/wiki/Tkinter.

The *tkinter* module is in the Python standard library and needs no installation. We'll introduce you to the basics of *tkinter*, including how to set up a screen and how to create a label widget.

The following program sets up a screen and adds a label to it. Run the following program *TkLabel.py*, as shown in Listing 6-3, in your Spyder editor.

```
#import all functions from the module
from tkinter import *

# create the root window
root = Tk()   #❶
#specify the title and size of the root window
root.title("A Label Inside A Root Window")   #❷
root.geometry("800x200")   #❸
# create a label inside the root window
label = Label(text="this is a label", fg="Red", font=("Helvetica", 80)) #❹
label.pack()   #❺
# run the game loop
root.mainloop()
```

Listing 6-3: Create a label in the tkinter *module*

We first import all functions from the *tkinter* module. Line ❶ sets up the top-level root window using the command `Tk()`. We name the root window *root*. The root window is used to contain all widgets created by a program.

Widgets are different types of small windows inside the top-level root window. However, they can also be stand-alone entities. There are many types of widgets such as buttons, labels, entries, and message boxes in the *tkinter* module. We will focus labels since we'll use them in the market watch projects.

Labels are a simple form of widgets. They are used to display messages or images for informational purposes.

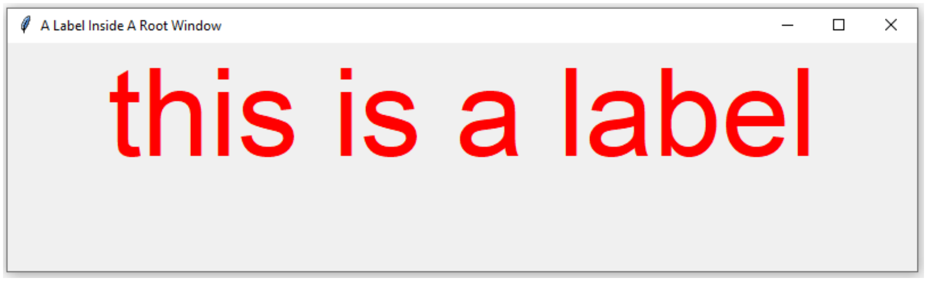
At ❷, we give the root window a title, "A Label Inside A Root Window". The `geometry()` method is called to specify the width and height of the root window to be 800 by 200 pixels ❸.

We initiate a label using the `Label()` function ❹. The main input in the function is the text (or image) you want to display. You can optionally specify the color and font of your message. In this example, we use red color with font set to `("Helvetica", 80)`.

The `pack()` method specifies where you want to put the label ❺. The default is to line up widgets starting from the top center of the root window. Finally, the `mainloop()` function starts the game loop so that the window shows up and stays on your computer screen.

If you run the program, you'll see an output as shown in Figure 6-3.

INSERT FIGURE 6-3 HERE



this is a label

Figure 6-3: A label inside the root window in tkinter

Create A Graphical Bitcoin Watch in tkinter

Next, you'll create a graphical Bitcoin watch using the *tkinter* module.

First, install the *arrow* library on your computer by activating your virtual environment and executing the following command in Anaconda Prompt (Windows) or a terminal (Mac or Linux)

```
pip install -U arrow
```

Open your Spyder editor and save the code in Listing 6-4 as *BitcoinTk.py* in your project folder.

```
# import the needed modules
from tkinter import *

import arrow     #❶
import requests
```

```
# specify the url to find the bitcoin price
url = 'https://api.coindesk.com/v1/bpi/currentprice.json'
# create a root window hold all widgets
root = Tk()   #❷
#specify the title and size of the root window
root.title("Bitcoin Watch")
root.geometry("1000x400")
# create a first label using hte Label() function
label = Label(text="", fg="Blue", font=("Helvetica", 80))  #❸
label.pack()
# create a second label
label2 = Label(text="", fg="Red", font=("Helvetica", 60))
label2.pack()
# define the BitcoinWatch() function
def BitcoinWatch():     #❹
    #get the live information from bitcoin url, put the price in m1
    response = requests.get(url)  #❺
    response_json = response.json()
    price=response_json['bpi']['USD']['rate_float']
    #obtain current date and time infomration
    tdate=arrow.now().format('MMMM DD, YYYY')  #❻
    tm=arrow.now().format('hh:mm:ss A')
    #put the date and time information in the first label
    label.configure(text=tdate+"\n"+tm)  #❼
    #put all the five messages on the stock market in the second label
    label2.configure(text=f'Bitcoin: {price}', justify=LEFT)
    #call the BitcoinWatch() function after 1000 milliseconds
    root.after(1000, BitcoinWatch)  #❽
# call the BitcoinWatch() function
BitcoinWatch()
# run the game loop
root.mainloop()
```

Listing 6-4: Create a graphical Bitcoin price watch

We import the necessary functions and modules first. In particular, we import the *arrow* module to show the current time and date ❶. We then use the `Tk()` method to create a top-level root window and specify the title and the size ❷. You need to pip install the *arrow* module before running the program.

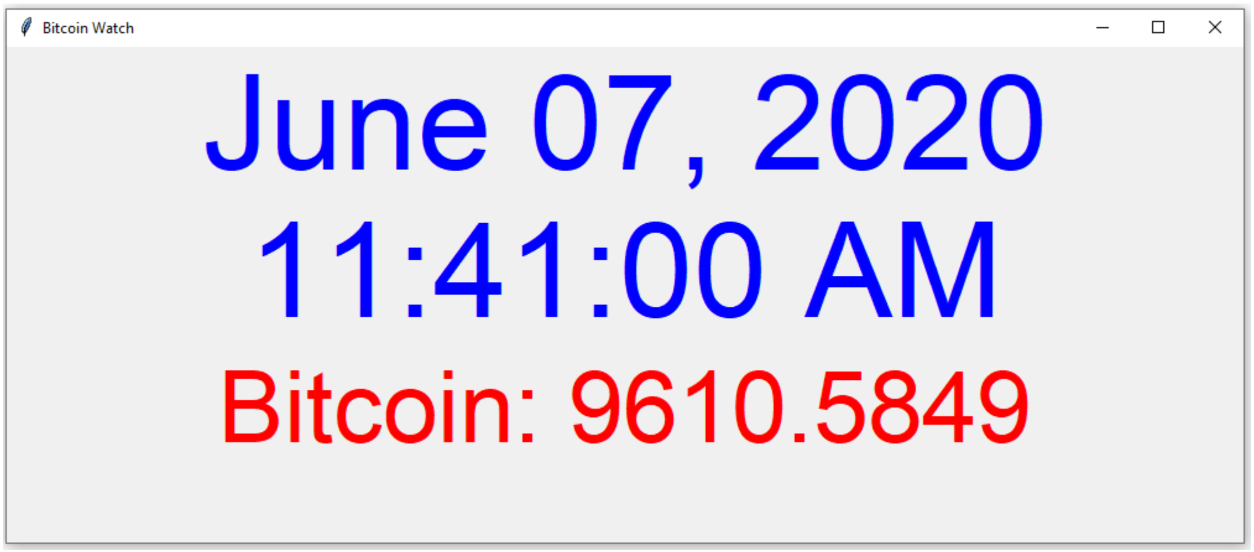
We create two labels using the `Label()` function ❸. We first leave the messages in both labels as an empty string because we'll adjust the message in them later. At ❹, we define the `BitcoinWatch()` function. The function first uses the *requests* module to obtain the Bitcoin price information from the url we have provided ❺. We also obtain the current date and time and save them in variables *tdate* and *tm*, respectively ❻.

At ❼, we put the current date and time information in the first label in separate lines using the escape character `\n`. We also put the live Bitcoin price in the second label in a different color, and the text is also left-justified.

Now let's discuss where the animation effect is created. The `after()` function is used to call another function after a specified amount of time. The command `after(1000, BitcoinWatch)` calls the function `BitcoinWatch()` after 1000 milliseconds ❽. Since the command is used within the `BitcoinWatch()` function itself, this creates an infinite loop in which all the command lines inside the `BitcoinWatch()` function are executed every 1000 milliseconds. If things change, we will see the animation effect. The result is that the time is constantly updated, and you can see the time value changes every second. If you keep the screen live long enough, you will also see the Bitcoin price change every minute or so.

If you run the above program, you will see a screen similar to Figure 6-4.

INSERT FIGURE 6-4 HERE


Bitcoin Watch

June 07, 2020
11:41:00 AM
Bitcoin: 9610.5849

Figure 6-4: Use the `after()` function to create an animated bitcoin watch

BEGIN BOX

Try it Out

6-2. Run the program *BitcoinTk.py* and watch it for about three minutes and see how often the price updates.

END BOX

A Graphical Live U.S. Stock Market Watch

Next you'll create a graphical live U.S. stock market watch. The program is based on the Bitcoin watch we just created. However, we'll make several significant changes.

First, instead of showing just one asset, we'll cover the major players in the market as well as the main assets that we are interested in. As a result, the program will show in a graph the values of two major indexes: the Dow Jones Industry Average and the S&P 500. It will also display the stock prices for the following three firms that we are interested in: Apple, Amazon, and Tesla.

Second, instead of updating every 1000 millisecond, we'll ask the program to update every two minutes. This is because the program needs to retrieve five pieces of information instead of just one piece of information. Updating too frequently will cause information overload in the program and lead to freezing of the Tk label. More importantly, the values for the market indexes and the prices for the above three stocks update every few seconds during the trading hours. Updating too often means the program will alert you non-stop and become distracting. Therefore, we choose to update the screen every two minutes. You can choose to adjust the frequency of updating in the program to your own liking.

Save the program in Listing 6-6 as *MarketWatch.py* on your computer (it's also in the zip file I put on Canvas).

```
# import needed modules
from tkinter import *

import arrow
from yahoo_fin import stock_info as si

# create a root window hold all widgets
```

```
root = Tk()  #❶
#specify the title and size of the root window
root.title("U.S. Stock Market Watch")
root.geometry("1100x750")
# create a first label using hte Label() function
label = Label(text="", fg="Blue", font=("Helvetica", 80))
label.pack()
# create a second label
label2 = Label(text="", fg="Red", font=("Helvetica", 60))
label2.pack()
# set up tickers and names
tickers = ['^DJI','^GSPC','AAPL','AMZN','TSLA']
names = ['DOW JONES','S&P500', 'Apple', 'Amazon', 'Tesla']
# set up the oldprice values and price bounds
oldprice=[]  #❷
maxprice=[]
minprice=[]
for i in range(5):
    p=round(float(si.get_live_price(tickers[i])),2)
    oldprice.append(p)
    maxprice.append(p*1.05)
    minprice.append(p*0.95)
    if i<=1:
        print(f'The latest value for {names[i]} is {p}!')
    else:
        print(f'The latest stock price for {names[i]} is ${p}!')
# define the StockWatch() function
def StockWatch():  #❸
    #declare global variables
    global oldprice
    #obtain live information about the DOW JONES index from Yahoo
    p1=round(float(si.get_live_price("^DJI")),2)  #  ❹
    m1=f'DOW JONES: {p1}'
    #obtain live information about the SP500 index from Yahoo
    p2=round(float(si.get_live_price("^GSPC")),2)
    m2=f'S&P500: {p2}'
    #obtain live price information for Apple stock from Yahoo
    p3=round(float(si.get_live_price("AAPL")),2)
    m3=f'Apple: ${p3}'
    #obtain live price information for Amazon stock from Yahoo
    p4=round(float(si.get_live_price("AMZN")),2)
    m4=f'Amazon: ${p4}'
    #obtain live price information for Tesla stock from Yahoo
    p5=round(float(si.get_live_price("TSLA")),2)
    m5=f'Tesla: ${p5}'
    # put the five prices in a list p
    p=[p1,p2,p3,p4,p5]  #⑤
    #obtain current date and time infomration
    tdate=arrow.now().format('MMMM DD, YYYY')
    tm=arrow.now().format('hh:mm:ss A')
    #put the date and time information in the first label
    label.configure(text=tdate+"\n"+tm)
    #put all the five messages on the stock market in the second label
    label2.configure(text=m1+"\n"+m2+"\n"+m3+"\n"+m4+"\n"+m5, justify=LEFT)
    # if there is update in the marekt, announce it
    for i in range(5):      #⑥
        if p[i]!=oldprice[i]:  #⑦
            oldprice[i]=p[i]
            if i<=1:
                print(f'The latest value for {names[i]} is {p[i]}!')
            else:
                print(f'The latest stock price for {names[i]} is ${p[i]}!')
    # if price goes out of bounds, announce it
    for i in range(5):       #⑧
        if p[i]>maxprice[i]:
            print(f'{names[i]} has moved above the upper bound!')
        if p[i]<minprice[i]:
            print(f'{names[i]} has moved below the lower bound!')
    #call the StockWatch() function after 5000 milliseconds
    root.after(120000, StockWatch)  #⑨
# call the StockWatch() function
StockWatch()
# run the game loop
root.mainloop()
```

Listing 6-6: Program to create a graphical live U.S. stock market watch

We first import needed modules. In particular, we import the *arrow* module to show the time and date and the *yahoo_finance* module to obtain stock price information.

Starting at ❶, we create the *tkinter* root window and place two labels in it, as we did in the program *BitcoinWatch.py*. We then create three lists *oldprice*, *maxprice*, and *minprice* ❷. We use the list *oldprice* to keep track of the values of the two indexes and the prices of the three stocks when we start running the program. The list *maxprice* are the five upper bounds, which are 5% above the corresponding values in the list *oldprice*. Similarly, we define the five lower bounds in the list *minprice*.

The program then announces the values of the two indexes and the prices of the three stocks. Note that we put a dollar sign ($) in front of the three stock prices but not in front of the two index values because index values are not measured in dollars.

A `StockWatch()` function is defined starting at ❸. The function first declares *oldprice* a global variable so that it can be used both inside and outside the function. Every time the function is called, it retrieves the values of two indexes and the prices of the three stocks from Yahoo Finance ❹. We keep two digits after decimal for all values and save them in a list *p* ❺.

We obtain the time and date and put them in the first label. We put the values of the two indexes and three stocks in the second label. At ❻, we check each of the five values. If any one of the values has an update ❼, we'll print it out. We also update the value stored in the list *oldprice* accordingly.

Starting at ❽, we check if any of the five values has gone out of bounds. If yes, the program makes an announcement. Finally, we use the `after()` function to create the animation effect ❾. The `StockWatch()` function calls itself every 120000 milliseconds (that is, two minutes). Therefore, the screen updates every two minutes.

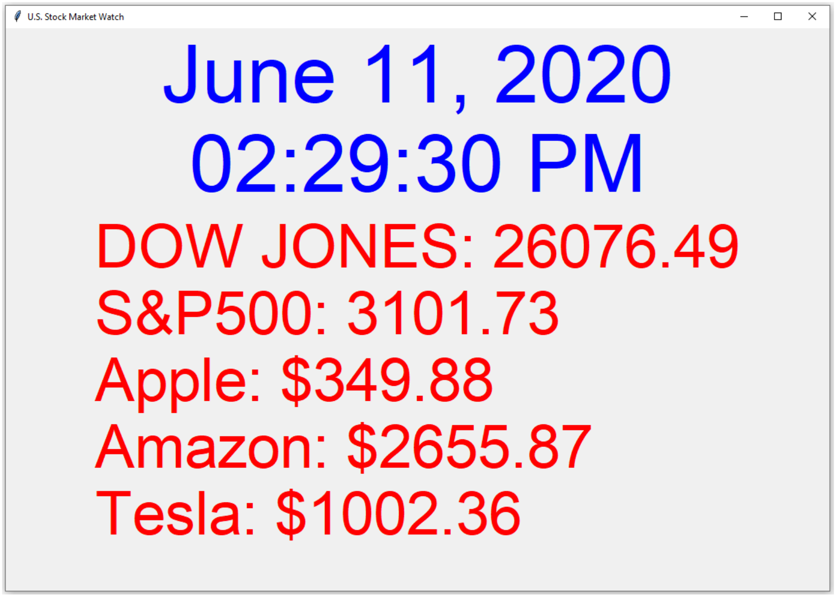
The output below is one interaction with the program.

```
The latest value for DOW JONES is 26069.84!
The latest value for S&P500 is 3101.06!
The latest stock price for Apple is $349.7!
The latest stock price for Amazon is $2655.45!
The latest stock price for Tesla is $1003.83!
The latest value for DOW JONES is 26076.49!
The latest value for S&P500 is 3101.73!
The latest stock price for Apple is $349.88!
The latest stock price for Amazon is $2655.87!
The latest stock price for Tesla is $1002.36!
The latest value for DOW JONES is 26099.64!
The latest value for S&P500 is 3104.5!
The latest stock price for Apple is $350.17!
The latest stock price for Amazon is $2662.59!
The latest stock price for Tesla is $1007.63!
```

In less than a minute, the program has updated all five values three times.

The generated graphical U.S. stock market watch is shown in Figure 6-5.

INSERT FIGURE 6-5 HERE



June 11, 2020
02:29:30 PM
DOW JONES: 26076.49
S&P500: 3101.73
Apple: $349.88
Amazon: $2655.87
Tesla: $1002.36

Figure 6-5: A graphical live U.S. stock market watch

BEGIN BOX

Try it Out

6-3. Change the three stocks in the program *MarketWatch.py* to Microsoft (ticker symbol MSFT), Goldman Sachs (ticker symbol GS), and Delta Airlines (ticker symbol DAL). Run the program after the change.

END BOX

Apply the Method to Other Financial Markets

You can apply the method we used in the program *MarketWatch.py* to other financial markets.

If the price information is available in Finance Yahoo using Python, the modification is minimal: all we need to do is change the ticker symbols in the programs. For example, you can create a global stock market watch by displaying the values of the following five market indexes: the Financial Times Stock Exchange 100 Index (ticker symbol `^FSTE`), the Euronext 100 Index (ticker symbol `^N100`), the Nikkei 225 Index (ticker symbol `^N225`), the Hang Seng Index (ticker symbol `^HSI`), and the Shanghai Stock Exchange Composite Index (ticker symbol `000001.SS`). All you need to do is to replace the five ticker symbols.

If the price information is not available in Finance Yahoo using Python, you can search online and see if you can find a website that provides JSON data for the market and use the same method we used to retrieve the Bitcoin price.

BEGIN BOX

Try it Out

6-4. Modify the program *MarketWatch.py* to create a graphical watch for the Treasury Bonds Rates. The graph should display the following four rates: 13-Week Treasury Bill rate (ticker symbol ^IRX), Five-Year Treasury Bond rate (ticker symbol ^FVX), Ten-Year Treasury Bond rate (ticker symbol ^TNX), 30-Year Treasury Bond rate (ticker symbol ^TYX).

END BOX

Summary

In this chapter, you first learned how to retrieve information from JSON data. You also learned how to create a graphical Bitcoin watch using the *tkinter* module to display live price information.

With these skills, you learned how to create a graphical live market watch for the U.S. stock market. The program generates a graphical display of two major U.S. stock indexes and three stocks that you are interested in. When the prices change, the program lets you know. The program also alerts you if an index value or a stock price goes out of the pre-set bounds

You also learned how to apply the method to create a graphical market watch in other financial markets.

End of Chapter Exercises

1: Modify *BitcoinPrice.py* so that: i) You retrieve the price in British pounds instead of in U.S. dollars; ii) the price is a string variable instead of a float number.

2: Modify *TkLabel.py* so that: i) the size of the root window is 850 by 160 pixels; ii) the message in the label says, “here is your label”.

3: Modify *BitcoinTK.py* so that the screen refreshes every 0.8 second.

4: Change the three stocks in the program *MarketWatch.py* to Microsoft (ticker symbol MSFT), Goldman Sachs (ticker symbol GS), and Delta Airlines (ticker symbol DAL). Run the program after the change.

5: Modify the program *MarketWatch.py* to create a graphical watch for the Treasury Bonds Rates. The graph should display the following four rates: 13-Week Treasury Bill rate (ticker symbol ^IRX), Five-Year Treasury Bond rate (ticker symbol ^FVX), Ten-Year Treasury Bond rate (ticker symbol ^TNX), 30-Year Treasury Bond rate (ticker symbol ^TYX). You don’t need to create alters when the rates go beyond the upper or lower bounds. Just a graphical display of the four rates.