**Bitcoin Price Notifications Using Python**

Prof. Kaustubh A. Hiwarekar, Geetai P.Charde

SY-Electronics and Telecommunication

***Abstract — As we all know, Bitcoin price is inconstant. No one knows where it’s going to be at the end of the day. So, instead of constantly checking various sites for the latest updates, this paper presents an idea of sending an alert to the user using Python programming language as soon as the Bitcoin price get change. This helps the user to saves their time.***

**Keywords: - Bitcoin, Python programming language.**

# *Introduction*

Bitcoin price notification is a tool which is going to send an alert to the user by sending an email or creating a GUI of latest Bitcoin price. “Bitcoin Price Notifications Using Python” utility is developed in Python programming language with the help of “Jupyter Notebook”IDE.

There are some Python libraries used to build this utility.

*1.BeautifulSoup 2.requests 3.time 4.smtplib 5.ssl 6.email.mime 7. PySimpleGUI*

**“BeautifulSoup”:-** Beautiful Soup is a Python library. It is use for pulling data out of HTML and XML files. To provide idiomatic ways of navigating, searching, and modifying the parse tree, it works with parser. It saves programmers hours or days of work. These instructions represent all major features of Beautiful Soup 4. It supports the HTML parser included in Python’s standard library. A number of third-party Python parsers is also supported by it. It gets transforms into a complex tree of Python objects from complex HTML document.

**“Requests”**:- Requests is one of the most downloaded Python package. It allows you to send extremely easily HTTP/1.1 requests. Query strings to your URLs, or to form-encode your POST data there is no need to add it manually. Keep-alive and HTTP connection pooling are 100% automatic. We can add content like headers, form data, multipart files, and parameters via simple Python libraries with it. In the same way it also allows you to access the response data of Python in.

**“Time”**:- This module provides various time-related functions such as *time.asctime([t]), time.clock\_gettime(), time.ctime([secs]), time.gmtime([secs]),* *time.localtime([secs]),* etc. out of this we are going to use *time.sleep(secs).*

“time.sleep(secs)”:- Its function is to suspend execution of the calling thread for the given number of seconds. To indicate a more precise sleep time, the argument may be a floating point number. The actual suspension time may be less than that requested because any caught signal will terminate the *sleep().* Also, because of the scheduling of other activity in the system the suspension time may be longer than requested by an arbitrary amount.

Changed in version 3.5: Even if the sleep is interrupted by a signal, the function is now sleeps at least sec, except if the signal handler raises an exception.

**“smtplib”**:- The “smtplib” module defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon. It also provides certain classes out of which *class smtplib.SMTP\_SSL(host='', port=0,local\_hostname=None, keyfile=None, certfile=None, [timeout,* *]context=None, source\_address=None)* is used here.

An *SMTP\_SSL* instance behaves exactly as same as the instances of *SMTP. SMTP\_SSL* should be used for situations where SSL is required from the beginning of the connection. The local host is used, if host is not specified. The standard *SMTP-over-SSL* port (465) is used if *port* is zero. The optional arguments *local\_hostname*, timeout and *source\_address* have the same significance as they have in the SMTP class. *context* is also optional it can contain a *SSLContext* and allows configuring various aspects of the secure connection. a Legacy alternative to context are *keyfile* and *certfile* they can point to a PEM formatted private key and certificate chain file for the SSL connection.

**“ssl”**:- SSL stands for Secure Sockets Layer. It is designed to create secure connection between client and server. Secure means that connection is encrypted and therefore protected from bug. It also allows to validate server identity. OpenSSL library is used by this module. It is available on near about all modern Unix systems, Windows, Mac OS X, and probably additional platforms, as long as OpenSSL is installed on that platform.

To use the *SSLContext.wrap\_socket()* of an *SSLContext* instance to wrap sockets as SSLSocket objects is recommended by Python 3.2 and 2.7.9. *create\_default\_context()* is the helper functions which returns a new context with secure default settings.

“ssl.create\_default\_context()”:-Return a new *SSLContext* object with default settings for the given purpose. Usually ssl module represents a higher security level than when calling the *SSLContext* constructor directly and also setting are choosen by it.

To trust for certificate verification, *cafile, capath, cadata* represent optional CA certificates as in *SSLContext.load\_verify\_locations(). T*his function can choose to trust the system’s default CA certificates instead if all three are *None*.

**“*email.mime”***:- This module is part of the legacy (Compat32) email API. In the new API its functionality is partially replaced by the *contentmanager*, but in certain applications these classes may still be useful, even in non-legacy code.

By passing a file or some text to a parser, which parses the text and returns the root message object, we get a message object structure ordinarily. However we can also build individual *Message* objects by hand, or even build a complete message structure from scratch. In fact, we can also take an existing structure and add new *Message* objects, move them around, etc. For slicing-and-dicing MIME messages, this makes a very convenient interface.

We can manually create a new object structure by creating Message instances, adding attachments and all the appropriate headers. The *email* package provides some convenient subclasses to make things easier for MIME messages.

Here we have use *class email.mime.multipart.MIMEMultipart(\_subtype='mixed', boundary=None, \_subparts=None, \*, policy=compat32, \*\*\_params).* This class is an intermediate base class for MIME messages that are multipart. IT is also a subclass of *MIMEBase*. Optional \_subtype defaults to mix. But it can be used to specify the subtype of the message. To the message object, a Content-Type header of multipart/\_subtype will be added. A MIME-Version header will also be added.

For the payload, *\_subparts* is a sequence of initial subparts. To convert this sequence to a list is possible. We can always attach new subparts to the message by using the *Message.attach* method.

**“*PySimpleGUI*”:-**

PySimpleGUI is a Python package that enables to create GUIs. we specify our GUI window using a "layout" which contains widgets (they're called "Elements" in PySimpleGUI). Layout is used to create a window using one of the 4 supported frameworks to display and interact with your window. Supported frameworks include tkinter, Qt, WxPython, or Remi. The term "wrapper" is sometimes used for these kinds of packages.

# *Methodology/Experimental*

**Algorithm:**

**Step 1**: Start.

**Step 2:** Import the required libraries.

**Step 3:** Create a function to get the price.

**Step 4:** Get the URL.

**Step 5:** Make a request to the website.

**Step 6:** Parse the HTML.

**Step 7:** Find the current price of crypto currency.

**Step 8:** Return the text.

**Step 9:** Store the email addresses for the receiver and the sender and store the sender password.

**Step 10:** Create a function to send emails.

**Step 11:** Create a MIMEMultipart Object.

**Step 12:** Create the HTML for the message.

**Step 13:** Create a html MIMEText Object and attach it to the message.

**Step 14:** Create the secure socket layer (SSL) context object.

**Step 15:** Create the secure simple mail transfer Protocol (SMTP) connection.

**Step 16:** Login to the email and send the mail.

**Step 17:** Create a function to display price alert on GUI.

**Step 18:** Set the layout.

**Step 19:** Create the window.

**Step 20:** Create an event loop.

**Step 21:** Close the window.

**Step 22:** Create a function to send the alert on the mail.

**Step 23:** Create an infinite loop to continuously send/show the price.

**Step 24:** Check if the price has changed then update the price.

**Step 25:** Send the alert.

**Step 26:** Stop.

# *Results*

# 

# Fig.1. Notification on the user’s mail

# 

# Fig.2. Notification through GUI

# *Benefits*

1. There is need to check for Bitcoin price manually.
2. It saves time.
3. User gets automatically notified.
4. It will be more beneficial for new users in digital currency business.

# *Limitations*

# If the website(from where we get the Bitcoin price) gets change, the system gets fail.

# *Future Scope*

This project can be built in such a way that it can send the text message directly on the user phone.

# *Conclusion*

# This Bitcoin Notification can be efficiently used for sending an alert to user. The output will be in the form sending an automatic mail to the user or in the form of GUI. Both the outputs are used for notifying the user with the new Bitcoin value.

# *References*

* [**https://docs.python.org/3/library/urllib.request.html**](https://docs.python.org/3/library/urllib.request.html)
* [**https://www.crummy.com/software/BeautifulSoup/bs4/doc/**](https://www.crummy.com/software/BeautifulSoup/bs4/doc/)
* [**https://docs.python.org/3/library/ssl.html**](https://docs.python.org/3/library/ssl.html)
* [**https://docs.python.org/3/library/smtplib.html**](https://docs.python.org/3/library/smtplib.html)
* [**https://docs.python.org/3/library/time.html**](https://docs.python.org/3/library/time.html)