TCS — Blockchain Tutorial 4

# Explore Bitcoin Blockchain with API Endpoints

So far in the Blockchain Tutorial 1 you set up a simple MVC framework with which to implement your blockchain explorer code. In Blockchain Tutorial 2 you explored more deeply the concept of the blockchain by writing some JavaScript code to simulate a simple blockchain, and you may have in Part 3 of this tutorial converted this JavaScript to a PHP implementation that integrates with your MVC framework. Further, in Blockchain Tutorial Part 4 you built a simple bitcoin currency exchange using three API endpoints from the Bitfinex, Bitstamp and Coinbase exchange APIs.

In this Blockchain Tutorial 4 we will investigate some more API endpoints to look at the bitcoin block data, starting with the [blockchain.info Transactions & Blocks Data API](https://blockchain.info/api). I haven’t got a blockchain app in mind at the moment, other than the notion of exploring the blockchain further. This Blockchain Tutorial 4 will simply access blockchain data using appropriate PHP methods called from their associated controller methods, which in turn push the returned blockchain data into the view via a JQuery handler. You have already built this MVC structure, so this tutorial will be sparse. I won’t give a line by line explanation of the code unless necessary. For this Blockchain Tutorial 4 we will use blockchain.inf Blockchain data API endpoints to query JSON data on bitcoin blocks and transactions to produce a view of the latest block in the bitcoin blockchain, which will something like that shown in Figure 1.



Figure 1: View of latest block header on the bitcoin blockchain

As a further exercise, you may also like to create a view of the transactions on latest block, Figure 2.

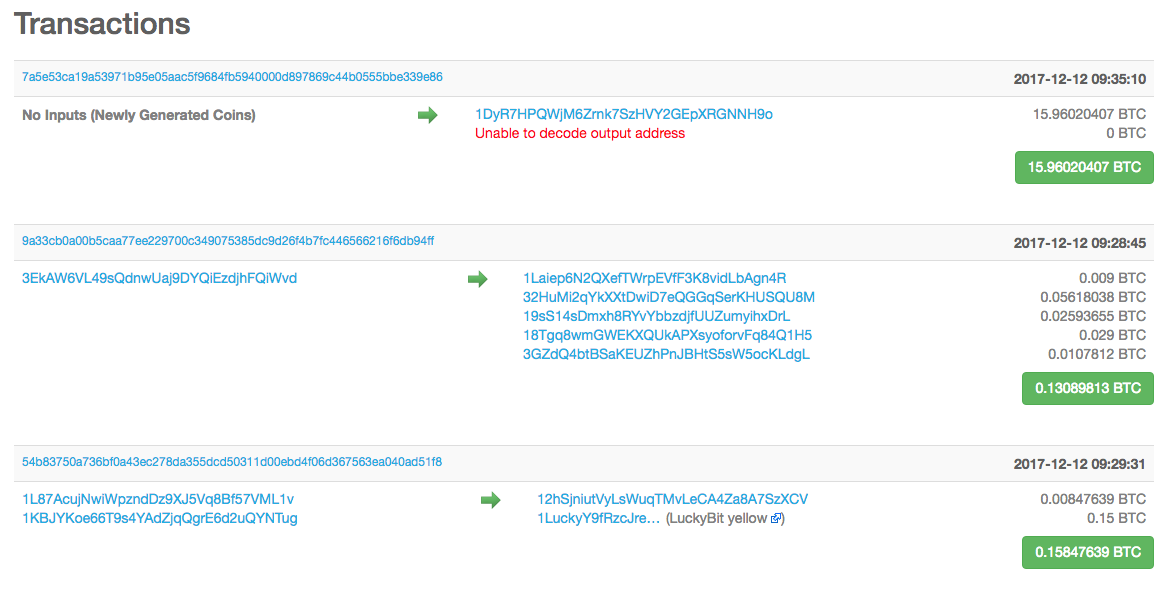


Figure 2: First three transaction on the latest block

You can see these latest block outputs on the [blockchain.info](https://blockchain.info/) web site, Figure 1 and Figure 2 have been cut and paste from a view of the Block #498883. To create our own view of the latest block we will use blockchain.info API endpoits.

Initially, we need to find:

* The **height** of the latest bitcoin block
* And, the **block\_index**, which will allow us to index the actual block.

This block data illustrated in Figure 1 and Figure 2 can be discovered by exploiting the following blockchain.inf Blockchain data API endpoints:

**Latest Block**: This API endpoint returns in JSON format the latest block hash (hash), its timestamp (time), the block index (block\_index), which index block in the chain, the height (height) of this latest block, and an array of transaction indexes (txIndexes) — I am not sure how to use the txIndexes at the moment, but they should be useful to, well, as it says on the tin, index a transaction on the block. If we execute this API call in a browser we can see the returned JSON format, Figure 3.

**Block Index**: From this ‘latestblock’ API endpoint we can find the blockchain height, although this doesn’t appear to be that interesting, other than a way of finding the block\_index, etc. More interestingly, we can index the latest block using the block\_index, Figure 4. We can see this is the same block (#498885), because the height values are identical. Incidentally, notice this is two blocks on from the block shown in Figure 1. This tells us that it took me about 20 mins between creating Figure 1 and Figure 3 because each block is added around every 10 minutes or so. The block\_index allows us to get more details on a specific block, as we can see from Figure 4. For blockchain.info, the block\_index is a unique number. If we increment the block\_index in Figure 4 two or three times and run the API endpoint again in the browser, you will see quite clearly each block is chained with the previous hash (prev\_block) value, Figure 5.

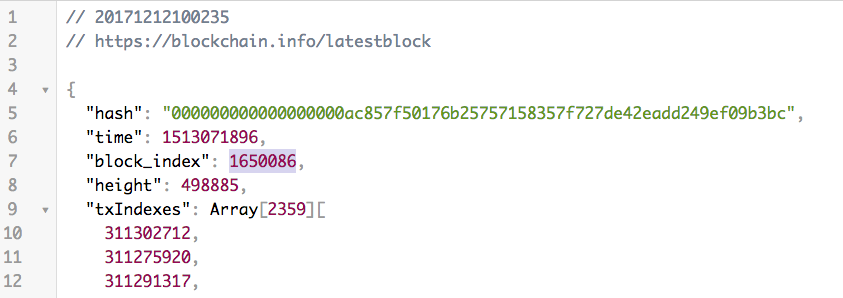


Figure 3: Find the latest bitcoin block using the blockchain.info ‘latestblock’ API endpoint

Further, Figure 4 shows us that we can now access all the information stored in the block header that we need to create the view shown in Figure 1. For example, we can see the:

* The **hash** of the last block,
* The previous hash, i.e. **prev\_block**
* The merkle\_root
* The timestamp, **time**
* The [difficulty target](https://en.bitcoin.it/wiki/Difficulty)[[1]](#footnote-1),[[2]](#footnote-2), **bits**, which is represented in a compact hexadecimal format, for this block. Here is another explanation on Stack exchange.
* The **nonce**, remember this was the random number found by the miner who solved the cryptographic hash to mine this block
* The miners **fee**, which is the sum of all the transactions fees
* And, lots of other information …

If you divide the miners fee in Figure 4 by 100 million, you get just over 2.2 BTC, which with 2,359 transactions in this block, works out at an average 0.00092 BTC per transaction fee for this block, which is about $4,74. However, transactions fees per transaction will vary, and has skyrocketed in recent days[[3]](#footnote-3). This article, ‘Bitcoin fees are skyrocketing’ explains quite nicely how bitcoin transaction fees work and discusses the issue around the bitcoin block size limit of 1 Mbyte. This size limits the average number of transactions in a block to something just less than 2,500 transactions, which works out at about 4 transactions per second based on the 10-minute time to add a new block to the blockchain. A miner will obviously choose to mine those transactions with a higher fee first. Rising demand is currently splitting the bitcoin community with those advocating for a change to the block size in order to increase transaction throughput against those advocating other methods. Bitcoin Cash, for example, is a fork of the mainstream Bitcoin software that allows blocks up to 8 Mbytes.

The miner will also receive 12.5 BTC as a reward, i.e. the ‘coinbase’. Note that a transaction may not have a transaction fee, because this is set by the user, e.g. Alice transferring her BTC to Bob. If Alice does not specify a transaction fee, then her transaction is likely to not get added to the blockchain at all. If your transaction is not very important you might specify a relatively low transaction fee and be prepare to wait hours or days tends to be included in a block.

You may recall that we discussed all this in the accompanying seminars, and in the Blockchain Tutorial 2 you built a very simple blockchain in JavaScript to better understand the operation of the blockchain.

So, in order to create a view of the latest block in the blockchain we can get most, if not all, the data from two blockchain.info API endpoint calls:

* **Latest block**: <https://blockchain.info/latestblock>
* **Block index**: [https://blockchain.info/block-index/1650086?format=json](https://blockchain.info/block-index/1650087?format=json)

Other vendors will offer similar API endpoints too.

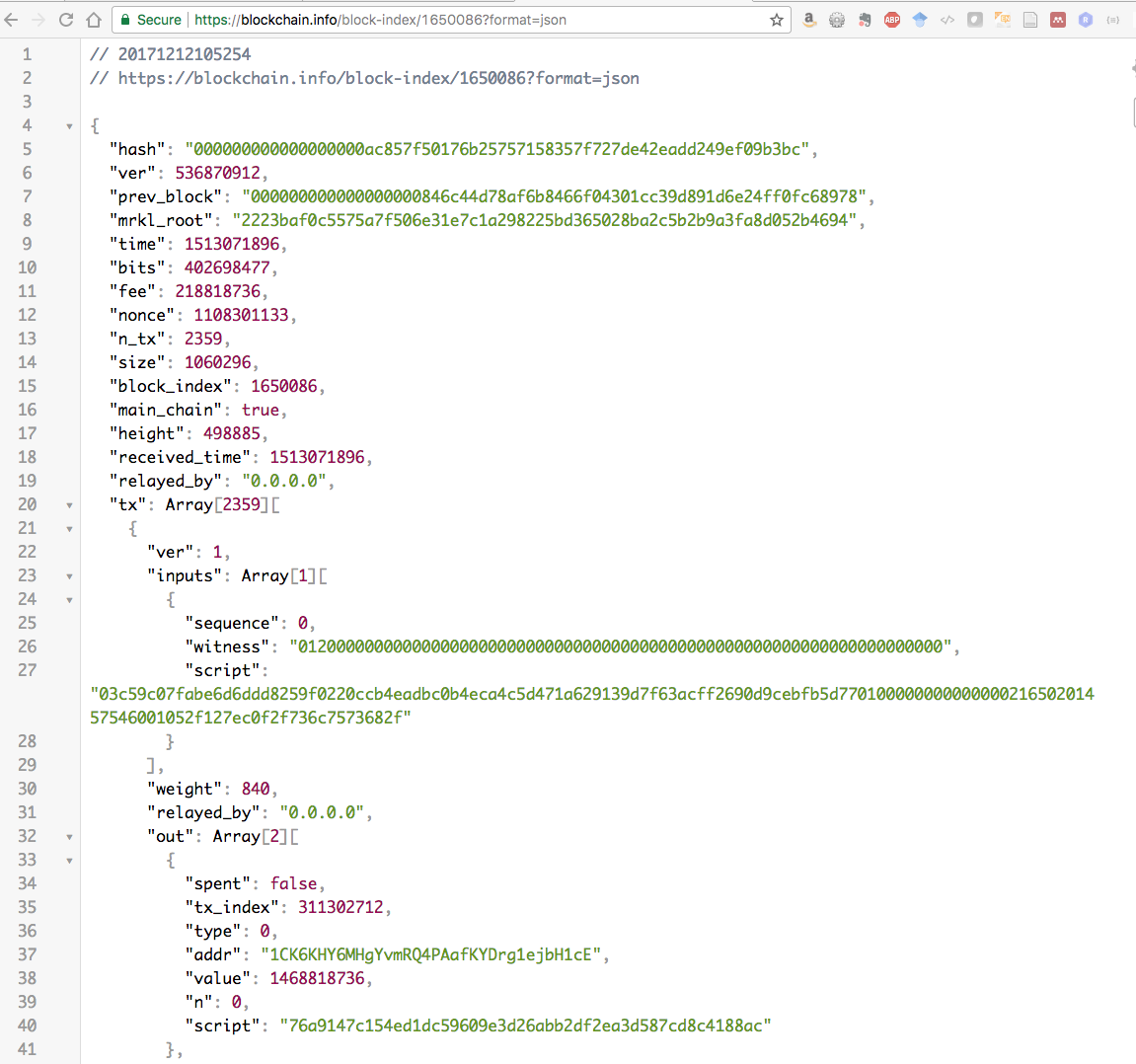


Figure 4: Using the block\_index to index the last block

However, what if we want to dig a little deeper and find out other information? For example, we might want to discover the BTC value of the miner who mined the latest block. We can do this by getting the address of the miner who received the bitcoin reward. It is always the first transaction in the block transaction array that transfers the block bitcoin reward (currently 12.5 BTC) plus the transaction fee to the miner address.

To find this address look at first transaction in the transaction array in the latest block, which in Figure 4 had "height": 498885, with “block\_index”: 1650086. This miner address is called "addr" and it's stored within the "out" array, where "out" is stored within the "tx" array, Figure 4. You may recall our explanation of this in the accompanying seminar.

Once we have this hash160 address (i.e. this is a SHA-256 hash of the public key, hashed again with the RIPEMD160 hash function transformed in some way[[4]](#footnote-4)). The transformed address is: 1CK6KHY6MHgYvmRQ4PAafKYDrg1ejbH1cE in Figure 4. In Figure 6, we can see both the hash160 and the transformed address. We can run the API endpoint in a browser to return the total received (total\_received) Satoshi at this miner address.

* **Address**: <https://blockchain.info/address/1CK6KHY6MHgYvmRQ4PAafKYDrg1ejbH1cE?format=json>

Figure 6 shows the total bitcoins received, sent and final balance in Satoshi for this miner

Dividing the "final\_balance": 18725864795 Satoshi, by 100 million reveals that this miner has 187.26 BTC at the time of writing. That’s a lot of dollars at the current Coinbase exchange rate of $16,948.99.

You should note that blockchain.info have other Query APIs that can be used to get all sorts of **Real-Time** data from the bitcoin blockchain, e.g. getdifficulty, getblockcount, latesthash, …, and for **Address Lookups**, e.g. getreceivedbyaddress/Address, addressbalance/Address, …

For example, you could use:

* **addressbalance/Address**: <https://blockchain.info/q/addressbalance/1CK6KHY6MHgYvmRQ4PAafKYDrg1ejbH1cE>

to get the balance in Satoshi for the miner address found above, i.e. 17367857965 Satoshi, 17340272596, which is $173.68 — it looks like this miners BTC balance has gone down since I accessed this miner’s address this morning.

This brings me neatly onto another point. It’s probably a good idea when accessing such data, for whatever application you have in mind, to grab the time stamp at the same time! If you look at in Figure 1, we can see that the time stamp is 2017-12-12 09:35:10, if we check the actual timestamp in the corresponding JSON output, which is indexed by block\_index = 1650084, we will see that we have "time": 1513071310. If you stuff this into an online timestamp convertor we get back Tue, 12 Dec 2017 09:35:10, a match.

When you write your code, you might use the PHP date() method to convert the time stamp to a more readable format. Alternatively, and more likely, you might convert the Unix timestamp using the JavaScript date() and toString() methods. For example, you can try this out in w3schools or jsfiddle:

<script>

unixDate();

function unixDate() {

var time = '1513071310'

var date = new Date(time\*1000);

document.write(date.toString());

}

</script>

The key methods being the JavaScript Date() and toString() methods and you will get the time from the jsonObj returned to your handler form the controller. You will need to invoke the unixDate() function, and then you should get returned: Tue Dec 12 2017 09:35:10 GMT+0000 (GMT) returned.

|  |
| --- |
| ../../../../Desktop/Screen%20Shot%202017-12-12%20at%2010.24.38.png |
| ../../../../Desktop/Screen%20Shot%202017-12-12%20at%2010.24.04.png |
| ../../../../Desktop/Screen%20Shot%202017-12-12%20at%2010.21.47.png |
| ../../../../Desktop/Screen%20Shot%202017-12-12%20at%2010.29.38.png |

Figure 5: Illustration of the block\_index API to view the prev\_block linking the blockchain



Figure 6: Run the blockchain address API endpoint to find the miner of the latest block

Ok, let’s write some PHP code to replicate what we did manually above. That is, create a view similar to that shown in Figure 1, and also display the value of the bitcoins owned by the miner who mined the latest block.

## Create the PHP model methods

First we are going to create a PHP method called apiGetLatestBlockData() method so that we can create a view similar to that shown in Figure 1. Since I grabbed this screen shot this morning, obviously more blocks have been added, so we can also see the next block hash. We will also create a PHP method to get the last block miner information: apiGetLatestBlockMinerData(), and tack it onto the end of this view under a title Miner Information.

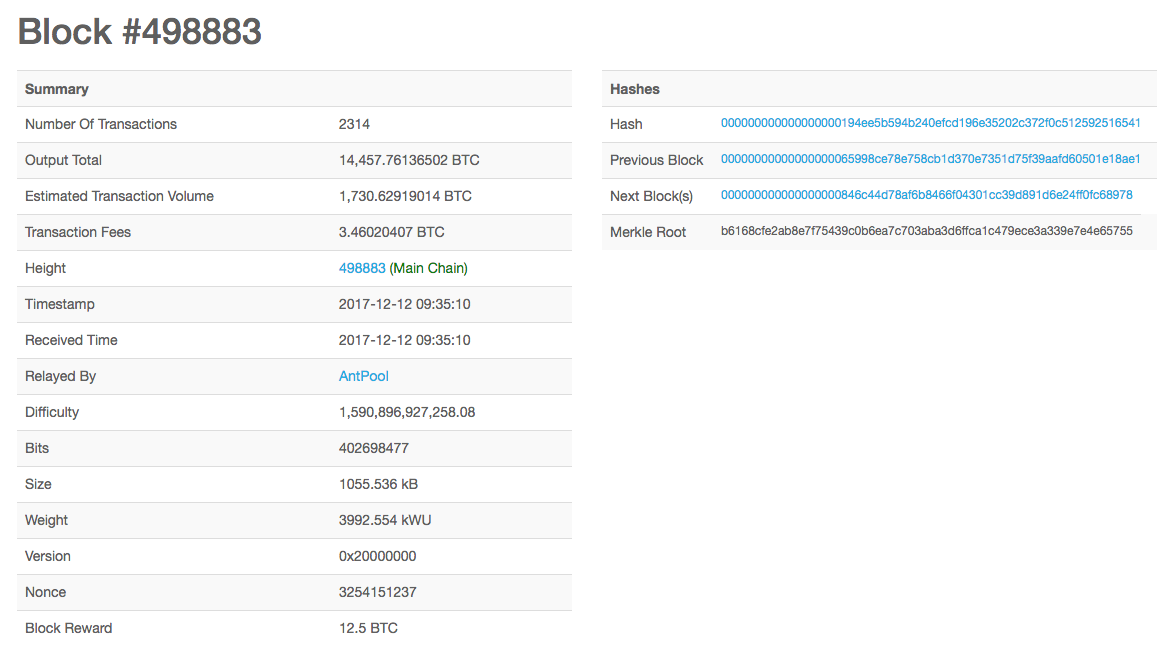


Figure 7: Mockup of the Block #498883 at timestamp 2017-12-12 09:35:10

That is about fifteen summary elements and 4 hashes that we need to grab for the Summary and Hashes. We will just grab the miner address along with the total\_received, total\_sent and final\_balance in Satoshi, but render in BTC, and give the dollar value. Figure 9 shows the PHD method, apiGetLatestBlockData(), to grab all this data using the Latest Block and Block Index API endpoints.

As already mentioned, I won’t bore you with a description of each line of code. You can see it is relatively straightforward and similar in principle to previous code you have seen. As before, we simply set up an array to hold the results obtained from invoking the two API endpoints:

* $latest\_block = "https://blockchain.info/latestblock?format=json";
* $block\_last = "https://blockchain.info/block-index/$block\_index?format=json";

We simply use the PHP file\_get\_contents() as before to turn the returned JSON format into a string, and then use PHP json\_decode() method to take this JSON encoded string into a PHP variable that we use to pick off each of the JSON name: value pairs in the bitcoin block header. We then simply stuff these in the result array to send back to the controller. As before, the controller will then use json\_encode() to package the result array backing into JSON format and echo it out. We could just work with the JSON format here, and echo it out, but this would bypass the controller. As before, you can simply cut and paste this code from the GitHub download.

Similarly, we can create the apiGetLatestBlockMinerData() method as described above, Figure 10. Again, we are using the latest block and block index API endpoints, but then we are digging into the first transaction because this stores information about the miner. Remember, we said the miner address is stored in the out array of the first transaction, which we can access with this piece of PHP code:

* $block\_miner\_address = $block\_json["tx"][0]["out"][0]["addr"];

Let’s look a bit more closely at Block #498883, which was the last block when I started writing this tutorial, Figure 1. The block index for this particular last block is: 1650084, which when invoked in the block-index API endpoint allows to look at the first transaction out array,

* https://blockchain.info/block-index/1650084?format=json



Figure 8: Miner transaction on the block

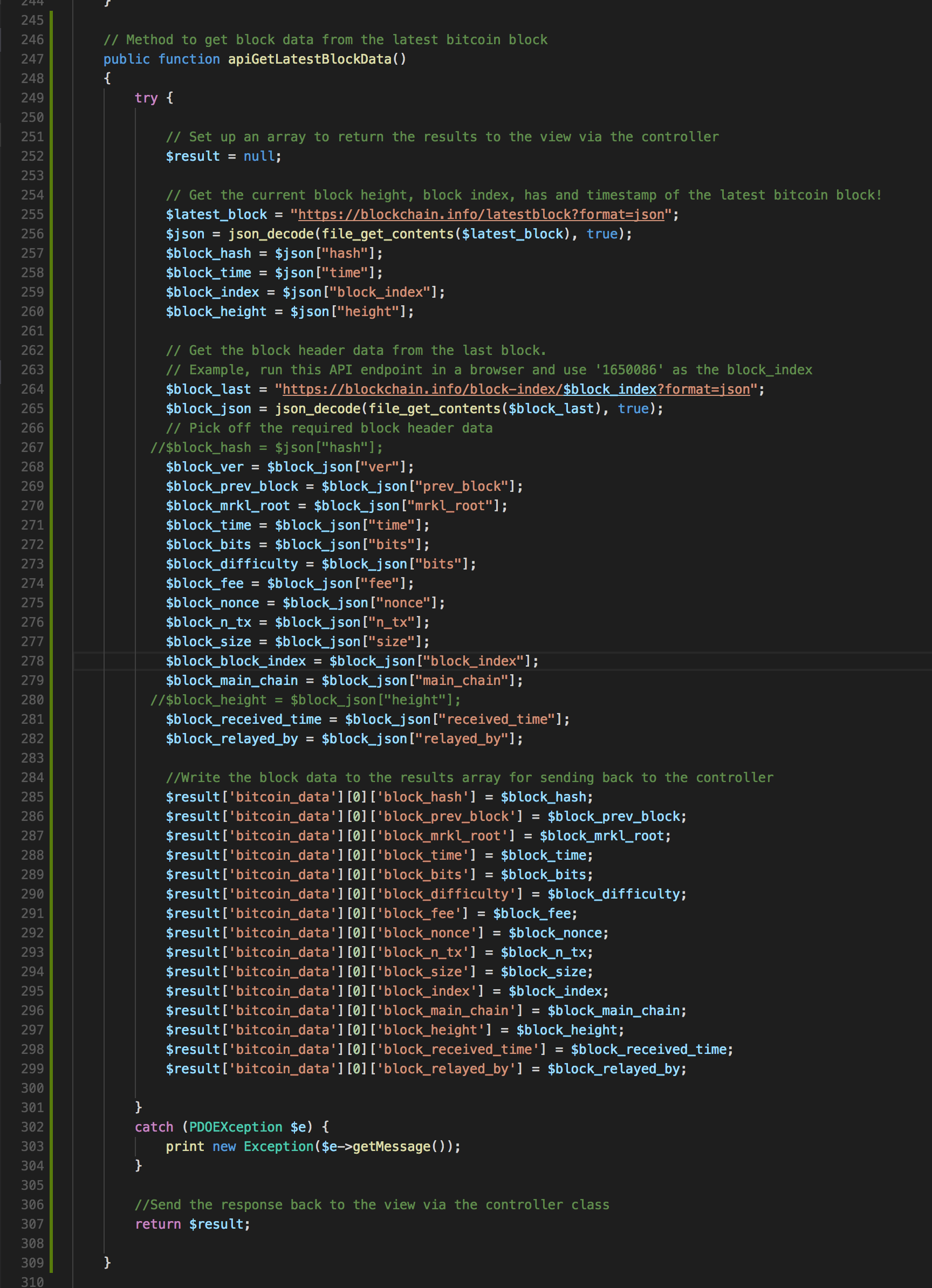


Figure 9: PHP method to get bitcoin latest block and miner data



Figure 10: The PHP method apiGetLatestBlockMinerData() used to extract miner data

As well as storing the miner address in this out array, Figure 8, we can see that the value, in Satoshi, of this transaction is also stored just below the miner address: "value": 1596020407. You can convert this Satoshi value to BTC by diving by 100 million. We can see that 1596020407/100000000 = 15.96020407 BTC. How is this? If we look at the fee for this block, we see that it is 346020407/100000000 = 3.46020407 BTC, which incidentally at an exchange rate of around $17,000 at the time of writing is not insignificant. If we subtract the fee of 3.46020407 BTC from the transaction value of 15.96020407 BTC we get the block reward of 12.5 BTC.

So, continuing with the PHP code in Figure 10, we can see that I have done these computations on the server side. We could just have easily done this in the client side in the getBitcoinData.js file in the jsonObj handler that you will need to create to stuff this data into the view.

We should test that these two new PHP methods work ok, and an easy way to do that is simply to write the associated PHP calling methods in the controller, Figure 11.

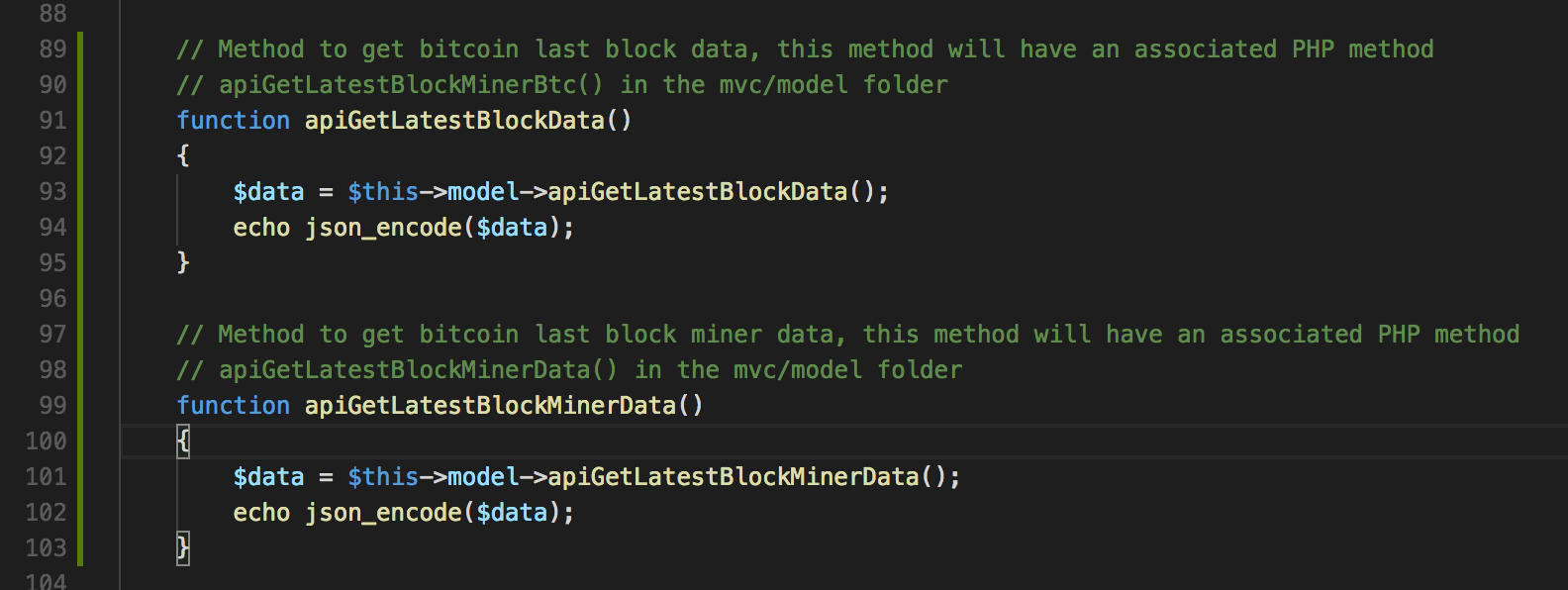


Figure 11: The associated PHP controller methods

We can easily test the model now by simply running these controller methods in a browser, Figure 12 and Figure 13. Note that the data returned in Figure 12 and Figure 13 is not from the Block #498883 — that block was accessed a couple of days ago. The block height has obviously increased since then.

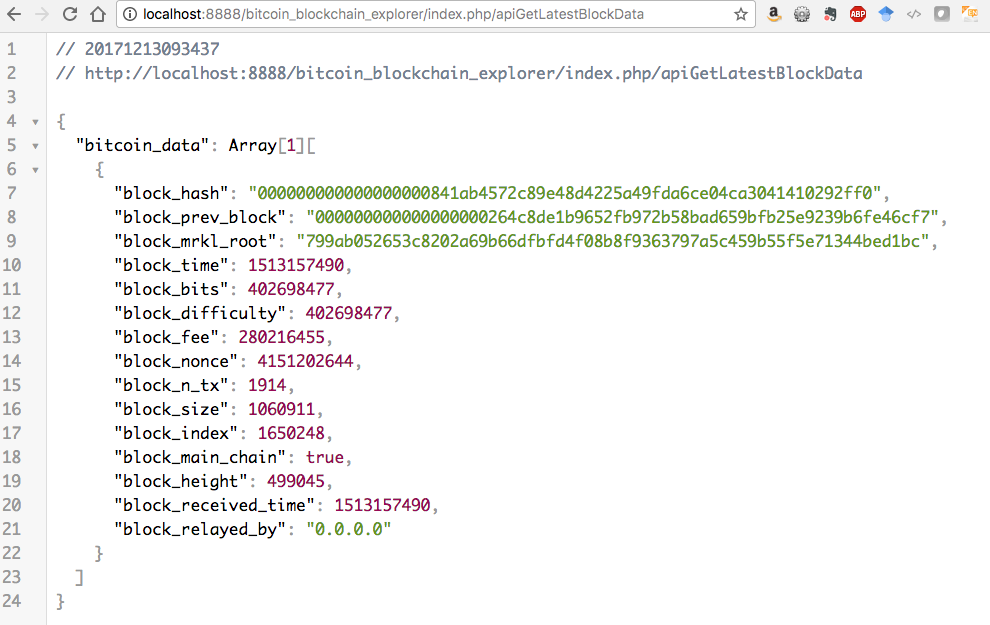


Figure 12: Test the apiGetLatestBlockData() controller method



Figure 13; Test the apiGetLatestBlockMinerData() controller method

Ok, so all that remains is for you to write the JavaScript code in the file getBitcoinData.js, which you can easily do based on existing code you have there. First, just a couple of paths to the newly created controller methods, url5 and url6 and invocations for two new JavaScript functions: apiGetBlockData() and apiGetBlockMinerData(), Figure 14.

Next create the two functions just mentioned; Figure 15 show two stubs for these JavaScript functions — the console.log() functions are sufficient to test that you are getting the jsonObj to the frontend view, Figure 16.



Figure 14: Add paths to the new controller methods



Figure 15: Stubs for the two JavaScript functions that call the controller PHP methods

You should be able to write the handlers based on code you already have. Have a go. Figure 16 shows an early example view that uses some of the values returned (from an early controller method). Check the values in this view can be found in the corresponding jsonObj in the Inspect panel in the Chrome browser.

To create a view similar to that the mock up shown in Figure 1, you will need to also modify the data.json file to provide appropriate metadata values, write the handlers in the JavaScript functions in Figure 14 and Figure 15, and replace the view in the bcExplorerBs.php file. Crack on.

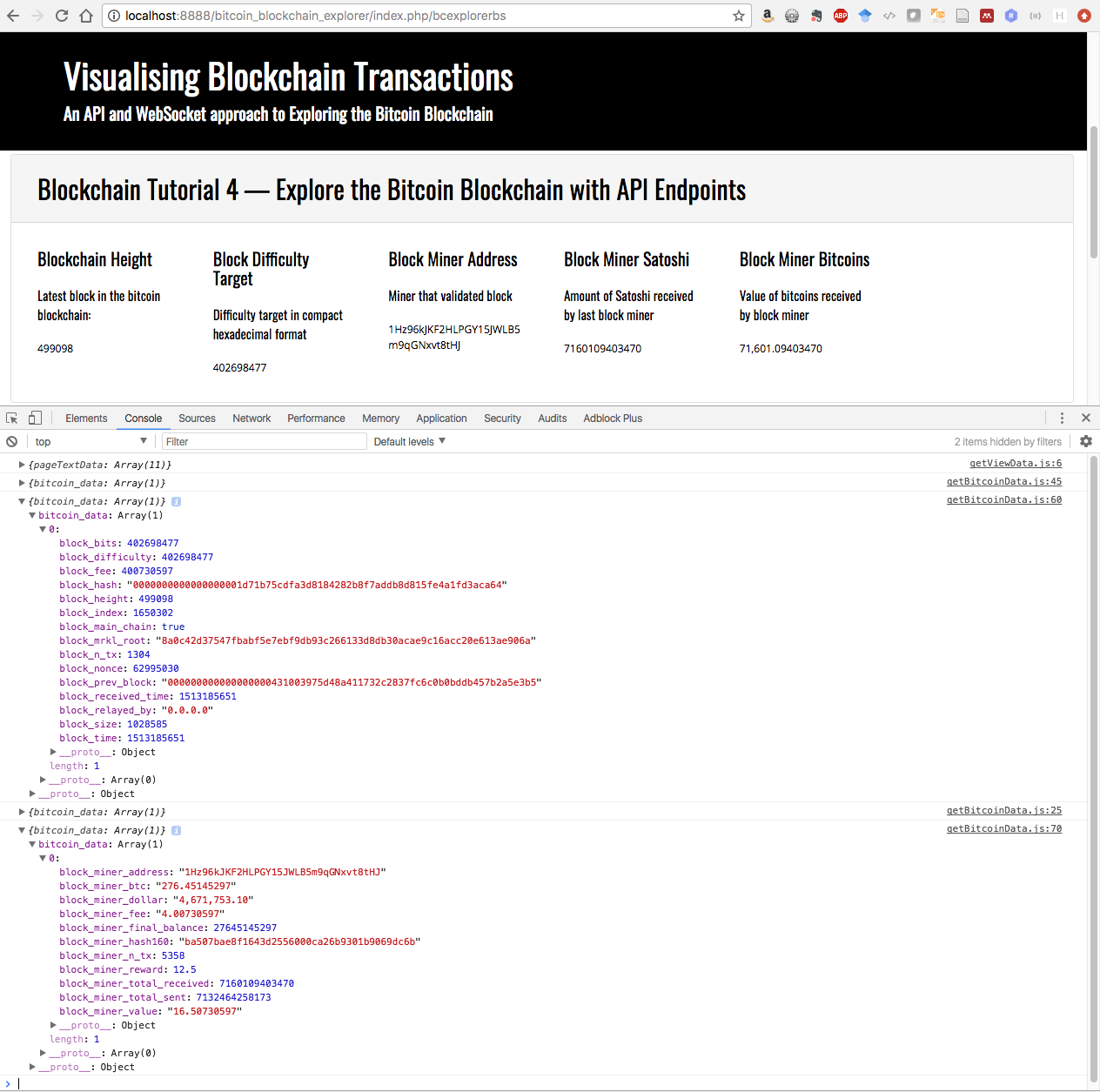


Figure 16: Early view of bitcoin block data and new block data returned from the controller

You should also, by now, be able to add the miner data to the view, and also consider how to add transaction data like that shown in Figure 2.

That concludes this Blockchain Tutorial 4. In the next Blockchain Tutorial 5, I plan to build a bitcoin price ticker using both API endpoints and also explore bitcoin block data in real-time using WebSockets.

1. <http://learnmeabitcoin.com/guide/difficulty> [↑](#footnote-ref-1)
2. <https://bitcoin.stackexchange.com/questions/30467/what-are-the-equations-to-convert-between-bits-and-difficulty> [↑](#footnote-ref-2)
3. <https://arstechnica.com/tech-policy/2017/12/bitcoin-fees-are-skyrocketing/> [↑](#footnote-ref-3)
4. Look this up. [↑](#footnote-ref-4)