

# Programming blockchain part 3: Python programming language

Guido van Rossum, a Dutch programmer, created Python in 1991. Python is based on a simple philosophy: Minimalist. One of the interesting things about Python is that it combines simplicity into a programming language by using spaces to denote code blocks instead of curly braces or keywords.

1. Programming blockchain part 1: C ++ programming language
2. Programming blockchain part 2: Javascript programming language

In the previous two sections you know why C ++ and JavaScript are chosen to program blockchain. In addition to these two programming languages, Python is also a popular choice in blockchain programming. This article will help you understand the reason behind it.

## Python programming language

Guido van Rossum, a Dutch programmer, created Python in 1991. Python is based on a simple philosophy: Minimalist. One of the interesting things about Python is that it combines simplicity into a programming language by using spaces to denote code blocks instead of curly braces or keywords. Let's see what this means.

1. How to install Python on Windows, macOS, Linux

Check out a simple 'hello world' program.

```
print ('Hello, world!')
```

Compare with C ++ 'hello world' program. It is much simpler.

So what about some more complicated things? Suppose we are adding two numbers and having results.

```
num1 = 1.5
num2 = 6.3
sum = float (num1) + float (num2)
print ('The sum of {0} and {1} is {2}'. format (num1, num2, sum))
```

The result will be: Sum of 1.5 and 6.3 with 7.8

So how to program an entire blockchain using Python? The following data and code are taken from the article written by Gerald Nash on Medium.

## Create blocks

First, create a block:

```
Block class:
def __init__ (self, index, timestamp, data, previous_hash):
self.index = index
self.timestamp = timestamp
self.data = data
self.previous_hash = previous_hash
self.hash = self.hash_block ()

def hash_block (self):
sha = hasher.sha256 ()
sha.update (str (self.index) +
str (self.timestamp) +
str (self.data) +
str (self.previous_hash))
return sha.hexdigest ()
```

### Analyze the code

We start by entering the **hash library** to use the **SHA 256** hash function (quite similar to Javascript).

Like JavaScript, blocks have the following values:

1. Index
2. Timestamp
3. Data
4. Previous hash
5. Hash.

Then assign hash values ??through a function, just like in Javascript.

### Create Genesis blocks

Now, create Genesis blocks:

Enter **datetime** as the current date.

```
def create_genesis_block ():
return Block (0, date.datetime.now (), 'Genesis Block', '0')
```

### Analyze the code

Enter datetime to set a timestamp.

The task now is to simply create the genesis blocks and manually assign it some data to operate. The previous hash value is '0' because it does not point to any other block.

### Create the rest of the blocks

Now, determine how the next blocks will be created.

```

def next_block (last_block):
    this_index = last_block.index + 1
    this_timestamp = date.datetime.now ()
    this_data = "Hey! I'm block" + str (this_index)
    this_hash = last_block.hash
    return Block (this_index, this_timestamp, this_data, this_hash)

```

### Analyze the code

So, how do we determine the values ??of every piece of data inside each block?

1. **The index block** is simply the index of the last block + 1.
2. **Timestamp** is the current date and time.

Data of the block is a simple message: '**Hey! I'm block**'.

Hash is being calculated using the previously called function.

And finally, all these values ??will be returned to the block.

### Create blockchain

Finally, create a blockchain.

```

blockchain = [create_genesis_block ()]
previous_block = blockchain [0]

num_of_blocks_to_add = 15

for i in range (0, num_of_blocks_to_add):
    block_to_add = next_block (previous_block)
    blockchain.append (block_to_add)
    previous_block = block_to_add
    # Tell everyone about it!
    print "Block # {} ?ã ???
c thêm vào blockchain!".format (block_to_add.index)
    print "Hash: {} n" .format (block_to_add.hash)

```

### Analyze the code

First, create the genesis block and set its value to '**previous\_block**'.

Then determine how many blocks are added. In this example, 15 blocks will be added.

Therefore, the extra block step will be repeated 15 times to add each block to the blockchain. Finally, the export of block numbers has been added to the blockchain through their index number, and the output of the hash is also exported.

This is the output:

```
Block #1 has been added to the blockchain!  
Hash: 6311e5906c3fcbdec077aeb4e3f15aba1a398ffbffe74cafb033163e83c65cb2  
  
Block #2 has been added to the blockchain!  
Hash: 45c714818a556cde8ddec533f903daa92acd4ea0a03518a30ae1c3522d0e81ae  
  
Block #3 has been added to the blockchain!  
Hash: b58f7644cd153a910eee8a3fd20e2a453bacd13d253e54cadace8d615d80bbae  
  
Block #4 has been added to the blockchain!  
Hash: 5048e4dc66538c6baf45c7a7f7a211b23b89612e9afb8a2be485a2849df58005  
  
Block #5 has been added to the blockchain!  
Hash: a3f2caa493bfacca51796c63ca7ab4214d6e4403d70751ba526846094adc0160  
  
Block #6 has been added to the blockchain!  
Hash: 78384b46b91f52616794edc297997b6317e961130424fb3d5787f2627d3f1308  
  
Block #7 has been added to the blockchain!  
Hash: 0ffe13c9e1ce084455bb5247bb850e82bb0ad93d3cf3b6d011e23e4f64b81f13  
  
Block #8 has been added to the blockchain!  
Hash: c52392c8a89cc20714c10d0811bfbb373fc16582df3d9a49bf2834a8160f6e7  
  
Block #9 has been added to the blockchain!  
Hash: be740eab1b04382df0a980f3e804fa46115fd5aa5587f631f8662a010f31af4f  
  
Block #10 has been added to the blockchain!  
Hash: 0a71c411f6a68561abb7899c82e9fa0b81566c7ba00a7384cf50359cd9783041  
  
Block #11 has been added to the blockchain!  
Hash: 298ba050170e8e9a0ffe5048b53c7543966ca38610d45ca8cd8e8266987b4b5e  
  
Block #12 has been added to the blockchain!  
Hash: 67716d6d2cbab68ea77e08d130dbefbf7ad61a352e97002ba759b1208f633cfc  
  
Block #13 has been added to the blockchain!  
Hash: 5825a74a677791368383428f8f6997d2af2b51dad9188cad35c40a340d0941  
  
Block #14 has been added to the blockchain!  
Hash: dcc4a9b103117ce1e95ef4f1eb4569a2562d85b1af20b44853fb4851bd65b6a9  
  
Block #15 has been added to the blockchain!  
Hash: 963d86d669d39b6cd02e84b8f70da8549c7f110fccebfd43d0d870204cdf90f
```

Obviously in both Python and Javascript, you can add more complex features like Proof Of Work. If you want to learn how to do that, you should look at Gerald Nash's article. But now, at least you know how to create a simple blockchain in Python.

See more:

1. Blockchain programming part 4: Java programming language

You finished reading the article "**Programming blockchain part 3: Python programming language**" edited by the [TipsMake](#) team. We hope this article has provided you with many useful tech tips and tricks. You can search for similar articles on tips and guides. Thank you for reading and for following us regularly.