



|             |  |
|-------------|--|
| Title       | GETTING STARTED WITH DECENTRALIZED APPS (DAPPS) USING PYTHON |
| Description | Intro  |
| Date        | June 19th, 2022  |
| Author      | Arashtad   |
| Author URI  | <a href="https://Arashtad.com">https://Arashtad.com</a>      |



If you have ever heard of blockchain, bitcoin, web3, etc, you are most probably familiar with the word Dapp. Dapps or Decentralized applications are web3-based apps that operate autonomously using smart contracts. A simple example of Dapp is a wallet that is connected to a blockchain like Ethereum and operates the transactions automatically without the need for any third-party organizations or central servers. In this article, we are going to get more familiar with the Dapps using Python.

## USING PYTHON FOR DAPPS: INSTALLING THE DEPENDENCIES

The first step for getting started with the development of decentralized applications in python is learning how to work with smart contracts using web3.py tools. With that being said, let's get started by installing web3.py on your operating system. The default operating system for this tutorial is Linux but we will mention the necessary installation guides for those who work with Windows or Mac OS. On Linux, make sure you have pip3 installed. To do so, we write:

```
pip3 --version
```

If you see a result like this:

```
pip 20.0.2 from /usr/lib/python3/dist-packages/pip (python 3.8)
```

Then, you have it installed. Otherwise in your terminal, write:

```
sudo apt install python3-pip
```

Now it's time to install web3, first of all, there are 2 dependencies to be installed:

```
pip install eth-tester web3  
pip install eth-tester[py-evm]
```

Finally, we can install web3:

```
pip3 install web3
```

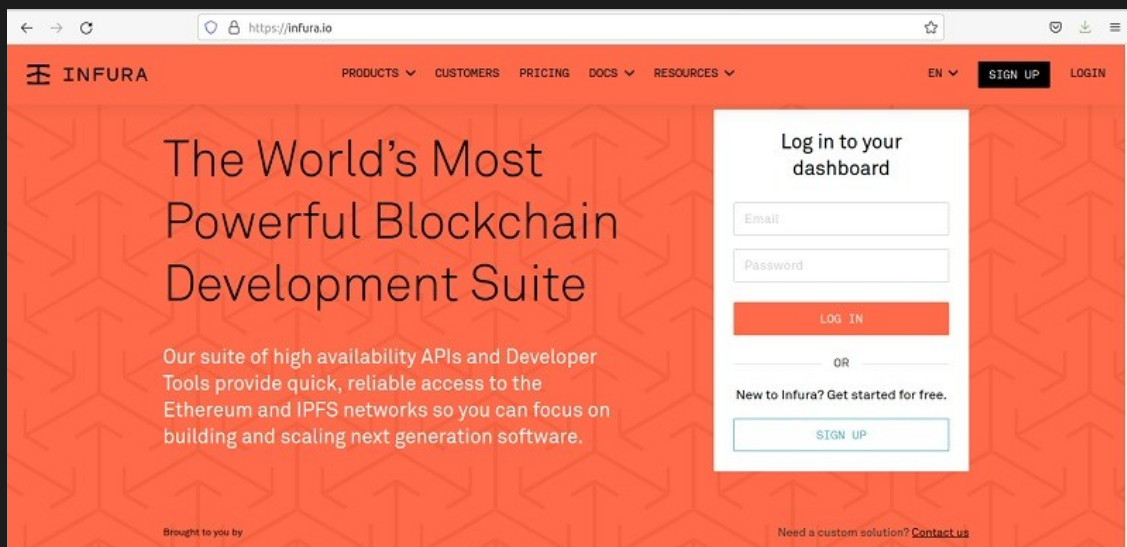
Notice that on windows instead of pip3, you should write pip and make sure you have installed pip on your operating system as well. You also need to have some Visual C++ dependencies installed when working on Windows. Otherwise, you will face some errors during the installation of web3.

After we have installed all the necessary packages, it is time to write our very first web3 scripts in python. Before we do that, we must connect to the Ethereum node. To do that, there are 2 options:

1. Running a local node on your computer using the Geth or Parity command: This method takes a lot of memory to download all the transactions on the Ethereum blockchain, not to mention the necessity to permanently connect to the internet to maintain a node, bandwidth conditions that should be met and so on.
2. Using a hosted node, like Infura or other similar ones.

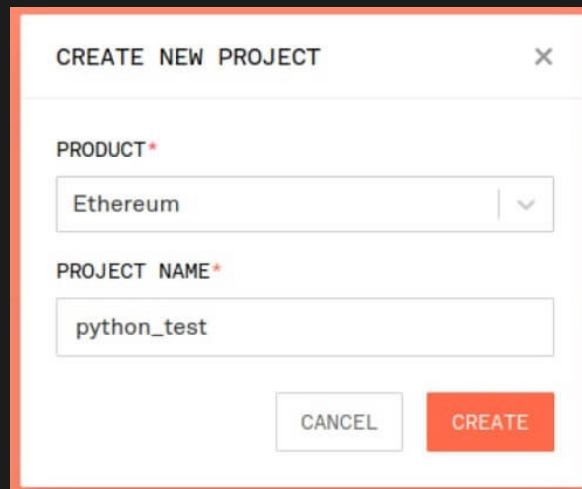
## INFURA

As we want to have our first experience with web3 in python, using Infura as a simple and free method is more preferred than other options. So, to do that you need to first sign up in infura.io:



The screenshot shows the Infura website's login page. The browser address bar displays "https://infura.io". The navigation menu includes "PRODUCTS", "CUSTOMERS", "PRICING", "DOCS", "RESOURCES", "EN", "SIGN UP", and "LOGIN". The main heading reads "The World's Most Powerful Blockchain Development Suite". Below this, a sub-heading states: "Our suite of high availability APIs and Developer Tools provide quick, reliable access to the Ethereum and IPFS networks so you can focus on building and scaling next generation software." On the right side, there is a login form with fields for "Email" and "Password", a "LOG IN" button, and a "SIGN UP" button. A link for "Need a custom solution? Contact us" is located at the bottom right.

And create a new project:



CREATE NEW PROJECT

PRODUCT\*

Ethereum

PROJECT NAME\*

python\_test

CANCEL CREATE

After creating a new project go to settings and in the key section copy the RPC URL which is in the following format:

`https://selected endpoint.infura.io/v3/Your Project ID`

It is recommended that you select Mainnet for your endpoint.

## PYTHON SCRIPTS FOR DAPPS

Now that we have our hosted node to be able to connect to Ethereum blockchain, it is time to get into some python scripts:

```
from web3 import Web3

infura_url = "https://mainnet.infura.io/v3/"
web3 = Web3(Web3.HTTPProvider(infura_url))
print(web3.isConnected())
print(web3.eth.blockNumber)
print(web3.eth.get_block('latest'))
```

Result:

```
True 14395685 AttributeDict({'baseFeePerGas': 27155659376,
'difficulty': 12673708419879581, 'extraData': HexBytes..... (to be
continued)
```

The above code checks whether we are connected to web3 or not, the number of blocks on the Ethereum blockchain and the latest block that is mined. Now, we go after some of the more useful functions like checking the validation of an account, its balance, tracking the transactions it has had and so on. To do that, we need an account address which we can get from etherscan.io. Then we will write:

```
validation =  
web3.isAddress(0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8)  
print(validation)
```

Result:

```
True
```

```
check_sum = web3.toChecksumAddress(accountAddress)  
balance = web3.eth.getBalance(check_sum)  
print(check_sum)  
print(balance)
```

Result:

```
0xa1bAA6C66930a3FB9803726f41D5E4F855805028 207816291093185845
```

Notice that if instead of `web3.eth.getBalance(check_sum)` you had written `web3.eth.getBalance(account)`, you would have face an error like this:

```
raise InvalidAddress( web3.exceptions.InvalidAddress: ('Web3.py only  
accepts checksum addresses. The software that gave you this non-  
checksum address should be considered unsafe, please file it as a  
bug on their platform. Try using an ENS name instead. Or, if you  
must accept lower safety, use  
Web3.toChecksumAddress(lower_case_address).',  
'0xa1baa6c66930a3fb9803726f41d5e4f855805028')
```



Also, remember the value that balance function returns is in Wei and needs to be changed to Ether, USD, or any other currency of your choice. If you want to change it to ether you can use the following code:

```
etherEquivalent = web3.fromWei(balance, 'ether')
```

Result:

```
0.200593459662878024
```

## CONVERSION TO US DOLLAR

If you want to convert the balance into a fiat currency like USD, EURO, or any other currency around the world, you need to get an API key from coin market cap website using this link. Sign up and copy your API key in the following script:



```
from requests import Request, Session
from requests.exceptions import ConnectionError, Timeout,
TooManyRedirects
import json
url =
'https://pro-api.coinmarketcap.com/v1/cryptocurrency/listings/latest
'

parameters = {
    'start':'2', # rank of the first token you want to fetch
    'limit':'2', # rank of the last token you want to fetch
    'convert': 'USD ' # You can write any other currency you want
to fetch your data in
}
headers = {
    'Accepts': 'application/json',
    'X-CMC_PRO_API_KEY': ' Paste your API key in here',
}

session = Session()
session.headers.update(headers)

try:
    response = session.get(url, params=parameters)
    data = json.loads(response.text)
    print(data['data'][0]['quote']['USD']['price']) #you can
specify the data that you want
                                                    #inside the
dictionary
except (ConnectionError, Timeout, TooManyRedirects) as e:
    print(e)
```

Result:

```
2590.1887169488728
```

Before running this code you need to consider the highlighted parts of the scripts. Here because we have wanted to convert Ethereum to USD, the start and limit variables are both 2 which means we want “our currency” equivalent of Ethereum, which here we have specified to be USD in the converted variable. Now, in order to have the USD equivalent of the balance, we use a simple multiplication:

```
USD_equivalent = data['data'][0]['quote']['USD']['price'] *  
etherEquivalent
```

Result:

```
519.5749159125255
```

Which shows that the account holds about 520 dollars in Ethereum.

The other useful web3 function is tracking the transactions. The following code returns different properties of a specific transaction:

```
transaction =  
  
web3.eth.get_transaction('0x77196fdfeb5b076f2fa4eb65fb59c5287010f171  
eef5635ef9c16905c4b2ebeb')  
  
print (transaction)
```



Result:

```
AttributeDict({'accessList': [], 'blockHash':  
HexBytes('0x84915d8a27c3127f9ac2417d44613a46c6e08dad34b1c5a16f493c8f  
4dc8d228'), 'blockNumber': 14395742, 'chainId': '0x1', 'from':  
'0xa1bAA6C66930a3FB9803726f41D5E4F855805028', 'gas': 108070,  
'gasPrice': 22953253192, 'hash':  
HexBytes('0x77196fdfeb5b076f2fa4eb65fb59c5287010f171eef5635ef9c16905  
c4b2ebeb'), 'input':  
'0xefbd73f4000000000...300000000000000000000000000000d3814659415aa213a2a  
76f5a9252a477b0e3f63', 'maxFeePerGas': 38902649594,  
'maxPriorityFeePerGas': 1000000000, 'nonce': 55, 'r':  
HexBytes('0x1caebee1505e611b5ae766af9339839bd1eaf2b0f8171cab49b1bd94  
e0362288'), 's':  
HexBytes('0x429a1717741dd4a153ae65d985a9ab2d230274b7311e47b46ba4193f  
5bcf7379'), 'to': '0x92069da581FC2f0705EaBEc9690779d23Ace6c3A',  
'transactionIndex': 102, 'type': '0x2', 'v': 0, 'value': 0})
```

As you can see, the result is a dictionary with some attributes. Here we can specifically print some of the more important ones such as “to”, “from”, or “gasPrice”.

```
print (transaction['to'])
```

Result:

```
0x92069da581FC2f0705EaBEc9690779d23Ace6c3A
```

```
print (transaction['from'])
```

Result:

```
0xa1bAA6C66930a3FB9803726f41D5E4F855805028
```

```
print (transaction['gasPrice'])
```

Result:

```
22953253192
```

There is also a 2nd way to look up for a transaction, which is using a transaction receipt:

```
transaction_receipt =
    web3.eth.get_transaction_receipt(
'0xd0f9e247581f9d4c5177fb315e7115e50fc9f673e0915b4b64f3ef5c1b8b81aa'
)
print(transaction_receipt)
```

Result:

```
AttributeDict({'blockHash':
HexBytes('0x166eff2ec3e1375ff70c1dd49b7e4e00dab4802f094fbf81d4021d6d
0ac48cb8'), 'blockNumber': 13557150, 'contractAddress': None,
'cumulativeGasUsed': 1719841, 'effectiveGasPrice': 270600000000,
'from': '0xDBD0C0C297035F3D9FD6788B6deC7A28dAd97C63', 'gasUsed':
47216, 'logs': [AttributeDict({'address':
'0xd665ce6Ef8AdA72B1CF946A6a71508bDD6D2EE04', 'blockHash':
HexBytes('0x166eff2ec3e1375ff70c1dd49b7e4e00dab4802f094fbf81d4021d6d
0ac48cb8'), 'blockNumber': 13557150, 'data':
'0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff',
'logIndex': 23, 'removed': False, 'topics':
[HexBytes('0x8c5be1e5ebec7d5bd14f71427d1e84f3dd0314c0f7b2291e5b200ac
8c7c3b925'),
HexBytes('0x00000000000000000000000000000000dbd0c0c297035f3d9fd6788b6dec7a28
dad97c63'),
HexBytes('0x00000000000000000000000000000007a250d5630b4cf539739df2c5dacb4c6
59f2488d')], 'transactionHash':
HexBytes('0xd0f9e247581f9d4c5177fb315e7115e50fc9f673e0915b4b64f3ef5c
1b8b81aa'), 'transactionIndex': 46})], 'logsBloom':
HexBytes('0x0000000000000000000000000000000000000000000000000000000000000000104..
.'), 'status': 1, 'to':
'0xd665ce6Ef8AdA72B1CF946A6a71508bDD6D2EE04', 'transactionHash':
HexBytes('0xd0f9e247581f9d4c5177fb315e7115e50fc9f673e0915b4b64f3ef5c
1b8b81aa'), 'transactionIndex': 46, 'type': '0x0'})
```



## IN SUMMARY

In this tutorial, we have got familiar with the python web3 tools and decentralized applications. In addition to that, we have managed to create an account on the Infura website to connect to it as a host node or HTTP provider. After connecting to the Ethereum Mainnet, we have checked the balance of an arbitrary account and using the coin market cap API, we have converted the price from Wei to US dollar.

