



Building Smart Contract Applications: Python, Solidity, & Flask

September 18, 2019

Michael Free



WatPy + Bitcoin Bay KW Meetup @ Terminal.io



Learning Outcomes

- Provide a Python “cheatsheet” for the upcoming ETHWaterloo 2 Hackathon in November
- Learn how to work with Ganache-CLI and Python to Perform Basic Ethereum Functions
- Build a Basic Storage Solidity Smart Contract
- Becoming familiar with Web3.py to Build a dApp
- Using Flask to Build a Web-based Python Application with Ethereum

Install Requirements

- GitHub Repository: <https://github.com/Michael-Free/PyDemo>
- Built on top of a vanilla Ubuntu Server 18.04 LTS install:

```
sudo apt install libz3-dev python3-dev python3-pip npm unzip
```

- Solc v0.4.25 is required:

```
wget https://github.com/ethereum/solidity/releases/download/v0.4.25/solidity-ubuntu-trusty.zip  
sudo unzip solidity-ubuntu-trusty.zip -d /usr/bin/  
rm -rvf solidity-ubuntu-trusty.zip
```

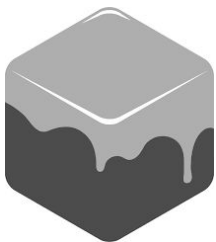
- This demo/tutorial uses Ganache-CLI and requires other libraries in requirements.txt

```
sudo npm install -g ganache-cli  
sudo pip3 install -r requirements.txt
```

Getting Started - Ganache-CLI

- An Ethereum Blockchain Emulator. Lightweight no need to run a node.
- When started with no parameters: 10 ETH addresses created
- Each address will have 100 ETH by default.
- Other information displayed:
 - HD Wallet mnemonic key: used to import these accounts into a wallet or other applications
 - Metamask, Parity, etc
 - Gas Limit and Gas Price:
 - Gas Limit - The amount of fuel is required to execute an operation or run a particular smart contract function.
 - Gas Price - Price set by the contract or the network, to execute the operation. This is variable. Choosing a lower gas price, means a lower-priority to execute the transaction (takes longer).
 - Transaction Cost = Gas Limit * Gas Price
 - Host address and port ganache-cli is listening on

Getting Started - Ganache-CLI



```
mike@pythondemo: ~/PyDemo 103x44
mike@pythondemo:~/PyDemo$ ganache-cli
Ganache CLI v6.4.5 (ganache-core: 2.5.7)

Available Accounts
=====
(0) 0x0d96a16809a4ec8d9c9a50483a4a1987d91e0130 (~100 ETH)
(1) 0x0dd94e12086d39669b8adb71ba4fd2bf295ec089 (~100 ETH)
(2) 0xe56ef8041539fb0c29040dd67157463cfbfe8475 (~100 ETH)
(3) 0xe84ebbc01327132c90ccaa299ab1016bc8a8b059 (~100 ETH)
(4) 0x9e6cd48cf8abf15d51ffd693790ecb684dd379ce (~100 ETH)
(5) 0x319eed767314107e9b8bb7417e4c1b616b1cee2b (~100 ETH)
(6) 0x3e7cf404e91ee330cff561a0a3056d1426815dd4 (~100 ETH)
(7) 0x127e32509583d02ca9aaf9b74fc0644952d04966 (~100 ETH)
(8) 0xbde3d00917b99b7db1ec7fd439288396aac95a34 (~100 ETH)
(9) 0xd3dcb17f0a97e365e787ea8abb5b8b46363da3f (~100 ETH)

Private Keys
=====
(0) 0x4ff68a7c443d8ba4f0d1c8211846cd125aa7c9f7dfa1752667f9f8508ca8d936
(1) 0x2908f94490fe8c60bc991732c6fdf7e68156d829928f8994ecd85cd1bcbd7850
(2) 0x73b4acf8525b4f9f140304b908917395a4976f6fa19844d770dc1b2b3f185e6c
(3) 0x9c9db02f6ac2700ac38bc31204124891a838a102efbad743bb60a99c5551c2da9
(4) 0x628f978619039c2b51e4068bd5aa3ae4e3074733ed79ad0198e2a24fb237957b
(5) 0x6c36463de1fcb8a85d663f31b85c59f8f1da8f9c6ae8629166f98c3b7eb83d71
(6) 0x876d989c12ef8d58b3964a04b12e3ed7a7f6b395f5d24b4d80df066b81c22f9a
(7) 0x9af512338045d8e36e5fb68c37da0a68140780db7934ce15c07f4f8a889f821f
(8) 0xba6a01151bacc64af1d81a28ec08fd7bdf0b4e19feab33a01b81de105606c3bc
(9) 0x3f08274f257b443b676fb131c54efdf79059f218c33b21d0d815d775a9350bec

HD Wallet
=====
Mnemonic:      lecture machine fiction install jewel stage forget away illegal decade lion assist
Base HD Path:  m/44'/60'/0'/0/{account_index}

Gas Price
=====
20000000000

Gas Limit
=====
6721975

Listening on 127.0.0.1:8545
```



Getting Started - Web3.py

- Web3 is an API to the Ethereum Blockchain to build applications.
- There are many implementations. The most widely used implementation is Web3.js, which Web3.py is derived from.
- Let's startup a python terminal and import the web3 libraries:

```
>>> from web3 import Web3, HTTPProvider
```

- Set Ganache-CLI as the Blockchain Provider:

```
>>> web3 = Web3(HTTPProvider('http://127.0.0.1:8545'))
```

Getting Started - Web3.py

- Let's see if we can find out the gas price:

```
>>> web3.eth.gasPrice
20000000000
```

- Let's see if we can get the balance of one of our ethereum accounts:

```
>>> web3.eth.getBalance('0xC599Ca9376b82651b8e9A2ee741B1b404dc41FF8')
1000000000000000000000
```

- Create a new account for yourself:

```
>>> web3.personal.newAccount('YOURPASSWORD')
'0xE972Dc8a9a0701A98dB8466FC555Bc10150Cd977'
```

- The new balance of that account is zero:

```
>>> web3.eth.getBalance('0xE972Dc8a9a0701A98dB8466FC555Bc10150Cd977')
0
```

Getting Started - Web3.py

- Interacting with Ganache-CLI with Python and Web3.py
 - Get a list of your personal accounts:

```
>>> web3.personal.listAccounts
```

- Notice that this lists the ethereum addresses started by ganache-cli.

```
>>> from web3 import Web3, HTTPProvider
>>> web3 = Web3(HTTPProvider('http://127.0.0.1:8545'))
>>> web3.personal.listAccounts
['0xf82580c1b9A15e74A850ccbfF67B55c1A05395d0', '0x6de4306310B7dc464a4de5feE44Fa0a27a212b67', '0x8Df8C0c661a5550C2e8102171Da6c5c1963860EA', '0xB30a12b5B6482F1b4C0b3E7A2BB2D54EBfd39084', '0x7ec745904006Dd0148E928e0bbBA365aB735Dae7', '0x074049fd576Ca29A30f500Da7f6d496dE6272Dea', '0x17403E84558B78f5c0e569cd1Ef1058f877E44F1', '0x76F04410715F7C8De6f24C7653b45D3f85B692EA', '0x99e28fEcc4C82A7BAD83Cb99B73CbEF7c9a78D99', '0xFdcD268Be30f156737759E72f1DC7A9fBeEAD2a1']
>>> []
```

- The call from python to Web3.py can be observed from the ganache-cli terminal as well.

```
Gas Limit
=====
6721975

Listening on 127.0.0.1:8545
personal_listAccounts
█
```


Getting Started - Web3.py

- Send some ETH to your new account from your other accounts:

```
>>> web3.eth.sendTransaction({'to': '0xE972Dc8a9a0701A98dB8466FC555Bc10150Cd977',  
                             'from': web3.eth.coinbase,  
                             'value': 1000000})  
HexBytes('0xdd1ca7444da4498e168954669e3f1381f0a3843c40adaaa8e55d32e07c7c5985')
```

- The response is the transaction hash registered on the blockchain. Here is the output of the transaction in ganache-cli:

```
eth_sendTransaction  
  
Transaction: 0xdd1ca7444da4498e168954669e3f1381f0a3843c40adaaa8e55d32e07c7c5985  
Gas usage: 21000  
Block Number: 1  
Block Time: Tue Aug 13 2019 23:41:23 GMT+0000 (UTC)
```

Explaining Smart Contracts

- Self-Executing contracts that exist on a blockchain
 - Think of it like a computer program
- Contracts can store terms between a buyer and a seller directly written into lines of code (solidity)
- Transactions with the contract are recorded on the blockchain.
- The goal is to provide fully self-executing and self-enforcing contracts, improving on our existing framework.

Explaining Smart Contracts

- **Moving Parts in the Next Step:**
 - **Solidity** - the smart contract language that is most commonly used. This is what this demo/tutorial will be using.
 - **Solc** - Solc is a binary and commandline interface for the Solidity Compiler (LLLC).
 - **LLLC** - the Lovely Little Language Compiler. This binary will translate Solidity Contracts into a Ethereum-Blockchain executable format.
 - **Py-Solc** - The python wrapper for the the solc binary.

Building a Smart Contract

```
pragma solidity ^0.4.21;
contract StorageContract {
    /* Define variable owner of the type address */
    string public serialnumber;
    address public assetowner;
    /* create an event for registration - events help return values for the ui. */
    event Registration(
        string serialnumber,
        address assetowner
    );
    /* create a function that uses the 2 variables */
    function setRegistration (string newSerialnumber, address newAssetowner) public {
        serialnumber = newSerialnumber;
        assetowner = newAssetowner;
        emit Registration(serialnumber, assetowner);
    }
}
```

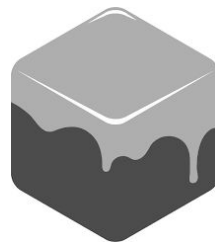
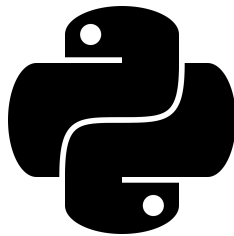
Learning More About Solidity

There are plenty of online resources for learning more about Solidity. For exploring more, take a look at some of the provided documentation and sample contract-implementations:

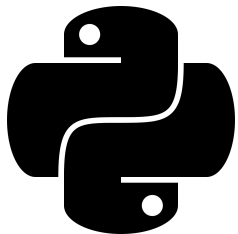
- **Solidity Documentation:** <https://solidity.readthedocs.io/en/v0.4.24/>
- **OpenZeppelin:** <https://github.com/OpenZeppelin/openzeppelin-contracts>
- **BlockGeeks:** <https://github.com/blockgeeks/workshop/tree/master/src/contracts>

Deploying Contracts (deploycontract.py)

- 2 ways to deploy a contract with Python:
 - Inline Code
 - What this demonstration will use.
 - Importing the Contract as a File
 - Not covered by this demo, but best practices provided in README.md



Deploying with Inline Solidity Code



```
from solc import compile_source
from web3 import Web3, HTTPProvider
from web3.contract import ConciseContract
web3 = Web3(HTTPProvider('http://127.0.0.1:8545'))
contract_source_code = '''
pragma solidity ^0.4.21;
contract StorageContract {
    /* Define variable owner of the type address */
    string public serialnumber;
    address public assetowner;

    /* create an event for registration - events help return values for the ui. */
    event Registration(
        string serialnumber,
        address assetowner
    );

    /* create a function that uses the 2 variables */
    function setRegistration (string newSerialnumber, address newAssetowner) public {
        serialnumber = newSerialnumber;
        assetowner = newAssetowner;
        emit Registration(serialnumber, assetowner);
    }
}
'''
```

deploycontract.py



Deploying Contracts

```
compiled_sol = compile_source(contract_source_code)
smartcontract_interface = compiled_sol['<stdin>:StorageContract']
StorageContract = web3.eth.contract(
    abi=smartcontract_interface['abi'],
    bytecode=smartcontract_interface['bin'])
web3.eth.defaultAccount = web3.eth.accounts[0]
tx_hash = StorageContract.constructor().transact()
tx_receipt = web3.eth.waitForTransactionReceipt(tx_hash)
assetregister = web3.eth.contract(
    address=tx_receipt.contractAddress,
    abi=smartcontract_interface['abi'],
)
```

deploycontract.py

Web³

Using Flask to Build a dApp - Libraries

Flask Requirements

```
# Flask requirements
from flask import Flask, render_template, jsonify, request, flash, redirect, url_for
from flask_bootstrap import Bootstrap
from flask_wtf import FlaskForm
from wtforms import StringField, SelectField, SelectField, validators
from wtforms.validators import InputRequired
```

dapp.py



Flask

Web3 Requirements

```
# DAPP Requirements
from hexbytes import HexBytes
from web3.auto import w3
from deploycontract import assetregister, StorageContract
```

dapp.py

Web³

Using Flask to Build a dApp - Input Form

Define an Input Form

```
# Registration Form for the application  
class RegisterForm(FlaskForm):  
    ethaddress = SelectField('Ethereum Address', choices=[])  
    serialnumber = StringField('Serial Number', [InputRequired()])
```

dapp.py



Flask

Using Flask to Build a dApp - App Routing

Application Routing: 3 Basic Routes

```
# Application routes
@app.route("/")
def home():
    return render_template(
        'home.html'
    )

@app.route("/register", methods=['GET'])
def register():
    return render_template(
        'register.html'
    )

@app.route("/registered", methods=['POST'])
def registered():
    return render_template(
        'registered.html'
    )

# Wrapper
```



Flask

dapp.py

Application Routing - home (/)

```
@app.route("/")
def home():
    return render_template(
        'home.html',
        contractaddress=assetregister.address
    )
```

dapp.py

HTML Templates - index.html

Basic template for application:

Variables can be passed to HTML templates.

The 'contractaddress' variable is inserted into the template with double curly brackets:

{{ contractaddress }}



```
<!DOCTYPE html>
<html>
<head>
<title>Bitcoin Bay KW + WatPy Solidity Demo</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">
<link rel="stylesheet" href="https://www.w3schools.com/lib/w3-theme-teal.css">
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
</head>
<body>
<div class="w3-container w3-padding-small w3-theme-d3">
  <div class="w3-right">
    Contract Address: {{ contractaddress }}
  </div>
</div>
<div class="w3-bar w3-theme w3-xlarge">
  <a class="w3-bar-item w3-button" href="/"><i class="fa fa-gears"></i></a>
  <span class="w3-bar-item">Bitcoin Bay + WatPy Solidity Demo</span>
  <a class="w3-bar-item w3-button w3-right" href="#"><i class="fa fa-search"></i></a>
</div>

{% block content %}{% endblock %}
</body>
</html>
```

All other routes will inherit this HTML template.

Their content will be shown between these 2 tags in `index.html`:

```
{% block content %}
{% endblock %}
```



templates/index.html

HTML Templates - home.html



This template is part of index.html

Nothing too interesting happening here.

Basically selecting the 1 menu option.

Just showing how to insert some code into a template.

```
{% extends 'index.html' %}
{% block content %}
  <div class="w3-cell-row" align="center">
    <div class="w3-container w3-cell w3-mobile">
      <div class="w3-card">
        <p><a href="/register"><i class="fa fa-address-card fa-5x"></i>
        <br>REGISTER</a></p>
      </div>
    </div>
  </div>
{% endblock content %}
```

Insert this HTML between these two tags in index.html:
{% block content %}
{% endblock %}



templates/home.html

What it looks like

```
mike@pythondemo: ~ - PyDemo
=====
Mnemonic:      toward umbrella peanut powder author survey exile craft clock shadow thrive friend
Base HD Path:  m/44'/60'/0'/0/{account_index}

=====
Gas Price
=====
20000000000

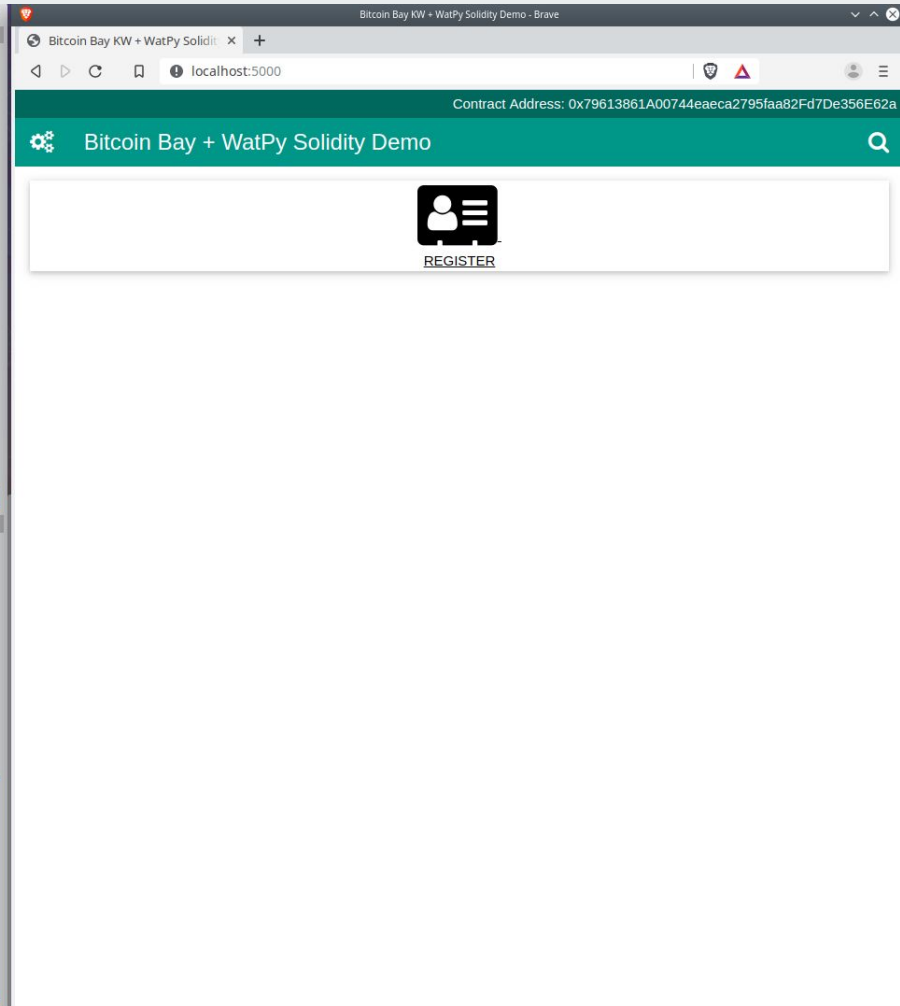
=====
Gas Limit
=====
6721975

Listening on 127.0.0.1:8545
eth_accounts
eth_estimateGas
eth_blockNumber
eth_getBlockByNumber
eth_sendTransaction

Transaction: 0xc39396712c13790d37cc87fc9beeafa36a64f1ab1eaa3f5a1e67766b3079273c
Contract created: 0x79613861a00744eaeca2795faa82fd7de356e62a
Gas usage: 380035
Block Number: 1
Block Time: Fri Sep 13 2019 01:48:22 GMT+0000 (UTC)

eth_getTransactionReceipt

mike@pythondemo: ~/PyDemo 103x28
mike@pythondemo:~/PyDemo$ export FLASK_APP="dapp.py"
mike@pythondemo:~/PyDemo$ export FLASK_ENV=development
mike@pythondemo:~/PyDemo$ export FLASK_DEBUG=0
mike@pythondemo:~/PyDemo$ flask run --host 0.0.0.0
 * Serving Flask app "dapp.py"
 * Environment: development
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
10.0.2.2 - - [13/Sep/2019 01:48:31] "GET / HTTP/1.1" 200 -
10.0.2.2 - - [13/Sep/2019 01:48:31] "GET /favicon.ico HTTP/1.1" 404 -
```



Application Routing - Register (/register) - GET

Call the form class created earlier.



n creates an index for each ethereum address.



Add +1 for each address created by ganache-cli (0-9).

```
@app.route("/register", methods=['GET'])
def register():
    form = RegisterForm()
    form.ethaddress.choices = []
    n = -1
    for chooseaccount in w3.personal.listAccounts:
        n = n+1
        form.ethaddress.choices += [(n, chooseaccount)]
    return render_template(
        'register.html',
        registerform=form,
        contractaddress=assetregister.address
    )
```

Return the form into the register.html template.

Return the contract address to the register.html template.



dapp.py

HTML Templates - register.html

```
{% extends 'index.html' %}
{% block content %}
  <div class="w3-cell-row" align="center">
    <div class="w3-container w3-cell w3-mobile">
      <p><i class="fa fa-address-card fa-5x"></i>
      <br>REGISTER</p>
      <form method="POST" action="{{ url_for('registered') }}" enctype="multipart/form-data">
        {{ registerform.csrf_token }}
        <table align="center">
          <tr>
            <td>{{ registerform.ethaddress.label }} :</td>
            <td>{{ registerform.ethaddress }}</td>
          </tr>
          <tr>
            <td>{{ registerform.serialnumber.label }} :</td>
            <td>{{ registerform.serialnumber }}</td>
          </tr>
        </table>
        <input type="submit" value="Register">
      </form>
    </div>
  </div>
{% endblock content %}
```

Call the ethereum addresses dropdown menu & label from the registerform class.



Call the serial number input field & label from the registerform class.



Import the index.html template

What it looks like

```
mike@pythondemo: ~/PyDemo
mike@pythondemo: ~ 103x27
Base HD Path: m/44'/60'/0'/0/{account_index}

Gas Price
=====
20000000000

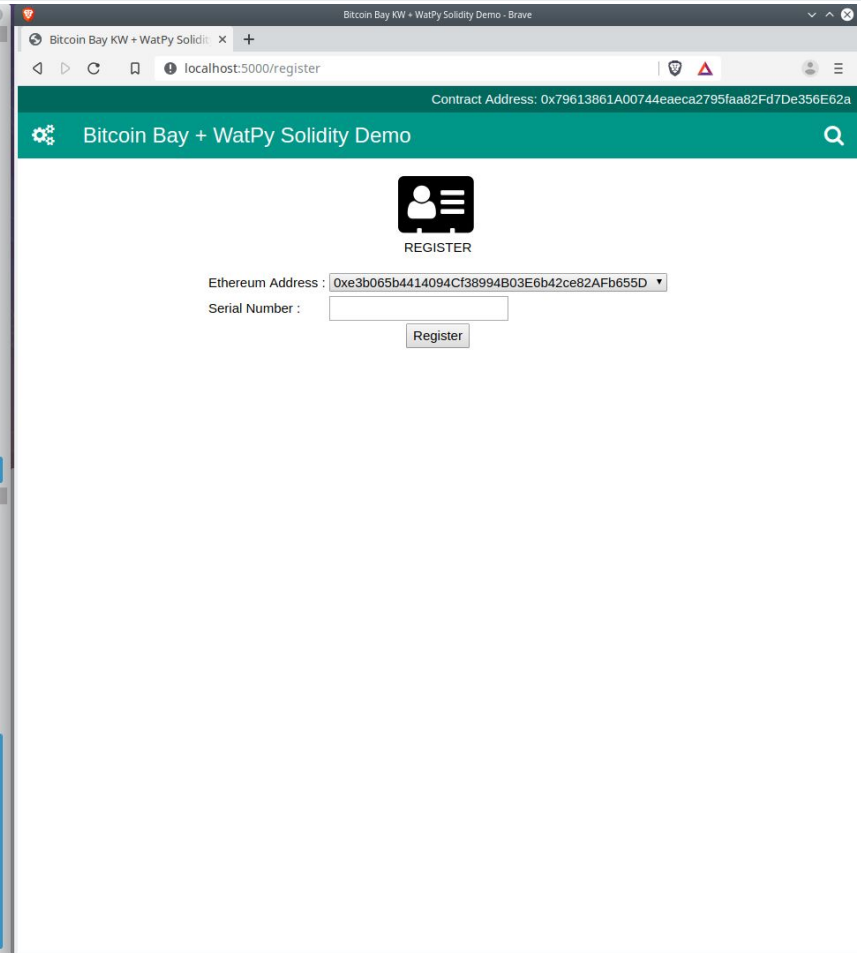
Gas Limit
=====
6721975

Listening on 127.0.0.1:8545
eth_accounts
eth_estimateGas
eth_blockNumber
eth_getBlockByNumber
eth_sendTransaction

Transaction: 0xc39396712c13790d37cc87fc9beeafa36a64f1ab1eaa3f5a1e67766b3079273c
Contract created: 0x79613861a00744eaeaca2795faa82fd7de356e62a
Gas usage: 380035
Block Number: 1
Block Time: Fri Sep 13 2019 01:48:22 GMT+0000 (UTC)

eth_getTransactionReceipt
web3_clientVersion
personal_listAccounts
]

mike@pythondemo: ~/PyDemo 103x28
mike@pythondemo:~/PyDemo$ export FLASK_APP="dapp.py"
mike@pythondemo:~/PyDemo$ export FLASK_ENV=development
mike@pythondemo:~/PyDemo$ export FLASK_DEBUG=0
mike@pythondemo:~/PyDemo$ flask run --host 0.0.0.0
 * Serving Flask app "dapp.py"
 * Environment: development
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
10.0.2.2 - - [13/Sep/2019 01:48:31] "GET / HTTP/1.1" 200 -
10.0.2.2 - - [13/Sep/2019 01:48:31] "GET /favicon.ico HTTP/1.1" 404 -
10.0.2.2 - - [13/Sep/2019 01:57:28] "GET /register HTTP/1.1" 200 -
]
```



Application Routing - Registered (/registered) - POST

Call the `setRegistration` contract function.

Pass the **address** and **serial number** string from the form to the contract

Print some of this info in the flask server window

Pass some variables to the templates. Not all are used - add them yourself!

```
@app.route("/registered", methods=['POST'])
def registered():
    registered = assetregister.functions.setRegistration(
        request.form['serialnumber'],
        w3.eth.accounts[int(request.form['ethaddress'])]).transact()
    tx = w3.eth.getTransaction(registered)
    tx_hash = HexBytes.hex(tx['hash'])
    print('TRANSACTION HASH:')
    print(str(tx_hash))
    print()
    tx_data = HexBytes(tx['input'])
    print('TRANSACTION DATA:')
    print(w3.toHex(tx_data))
    return render_template(
        'registered.html',
        reg_ethaddress=w3.eth.accounts[int(request.form['ethaddress'])],
        reg_serial=request.form['serialnumber'],
        reg_accountnumber=request.form['ethaddress'],
        reg_receipt=w3.eth.getTransactionReceipt(registered),
        reg_txhash= tx_hash,
        reg_txdata= tx_data,
        contractaddress=assetregister.address
    )
```

Create some vars to pass to the template from the contract transaction

dapp.py

HTML Templates - registered.html

```
{% extends 'index.html' %}
{% block content %}
    <div class="w3-cell-row" align="center">
        <div class="w3-container w3-cell w3-mobile">
            <p><i class="fa fa-address-card fa-5x"></i>
            <br>REGISTERED</p>
            Ethereum Address: {{ reg_ethaddress }} <br>
            Serial Number: {{ reg_serial }} <br>
        </div>
    </div>
{% endblock content %}
```

← Import the index.html
template

→ Call vars passed from the
dapp.py/registered route

templates/registered.html

What it looks like

```
mike@pythondemo: ~/PyDemo
eth_getBlockByNumber
eth_sendTransaction

Transaction: 0xf72c7501c5e793271c5666c8cba2267c5731c7872676479300b847a17332150a
Contract created: 0xba3084ceef5e4e2c50ef84be619b35ced1bc58ea
Gas usage: 380035
Block Number: 1
Block Time: Fri Sep 13 2019 02:12:00 GMT+0000 (UTC)

eth_getTransactionReceipt
web3_clientVersion
personal_listAccounts
eth_accounts
eth_estimateGas
eth_blockNumber
eth_getBlockByNumber
eth_sendTransaction

Transaction: 0x9acf142fb5e4ae580f9720e604830e0251814de51c9ddd885ecd1cda78909bc4
Gas usage: 67772
Block Number: 2
Block Time: Fri Sep 13 2019 02:12:41 GMT+0000 (UTC)

eth_getTransactionByHash
eth_accounts
eth_getTransactionReceipt
```


```
mike@pythondemo: ~/PyDemo 103x28
mike@pythondemo:~/PyDemo$ export FLASK_APP="dapp.py"
mike@pythondemo:~/PyDemo$ export FLASK_ENV=development
mike@pythondemo:~/PyDemo$ export FLASK_DEBUG=0
mike@pythondemo:~/PyDemo$ flask run --host 0.0.0.0
* Serving Flask app "dapp.py"
* Environment: development
* Debug mode: off
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
10.0.2.2 - - [13/Sep/2019 02:12:06] "GET / HTTP/1.1" 200 -
10.0.2.2 - - [13/Sep/2019 02:12:13] "GET /register HTTP/1.1" 200 -
10.0.2.2 - - [13/Sep/2019 02:12:41] "POST /registered HTTP/1.1" 200 -
```

Bitcoin Bay KW + WatPy Solidity Demo - Brave

localhost:5000/registered

Contract Address: 0xBA3084CeE5E4e2c50ef84Be619b35CeD1Bc58ea

Bitcoin Bay + WatPy Solidity Demo

 REGISTERED

Ethereum Address: 0x5CCC9BA630A0804d657144Dccff041122a96142B
Serial Number: ASDF123