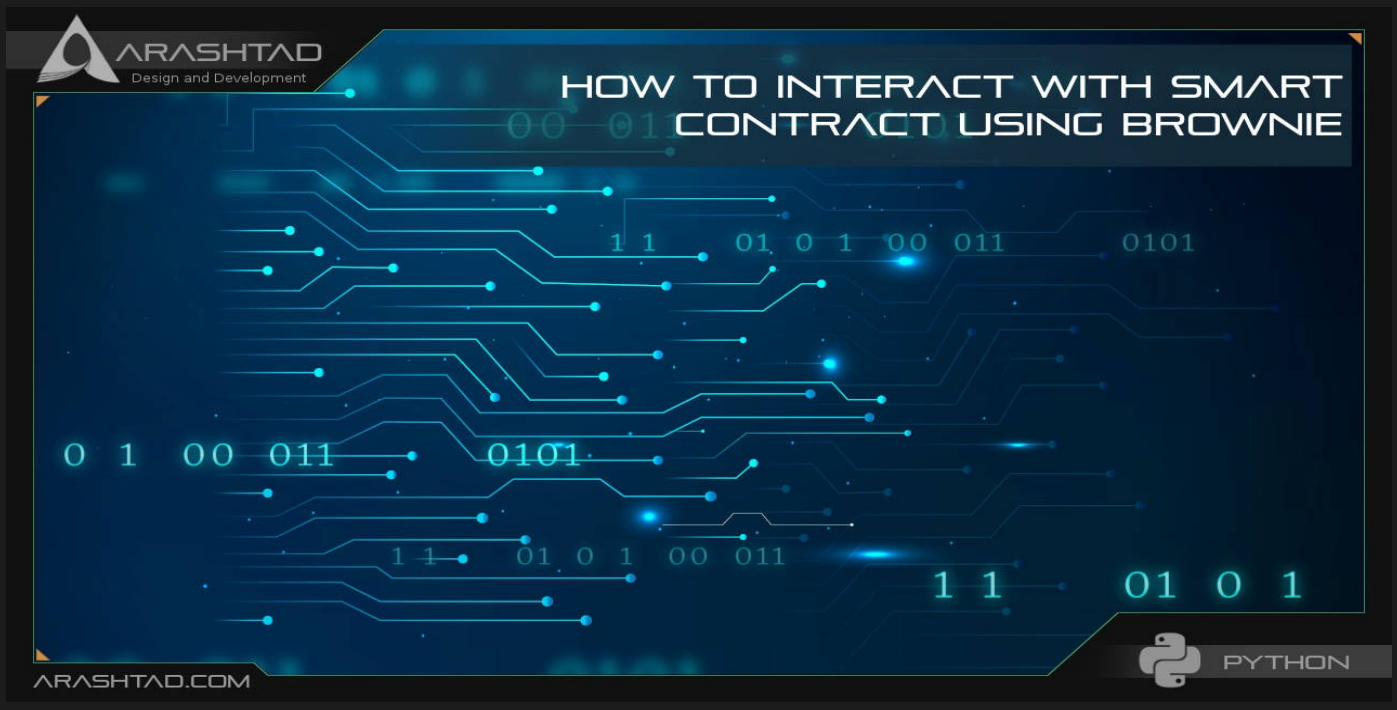
Title	HOW TO INTERACT WITH SMART CONTRACTS USING BROWNIE
Description	Intro
Date	June 12th, 2022
Author	Arashtad
Author URI	https://Arashtad.com



After installing and working with Ganache CLI, we are going to interact with simple storage smart contracts Using Brownie. Meaning that we want to store and retrieve a value within the contract. After doing this you will see how simple and efficient Brownie is compared to other methods in python. We are also going to use testing modules to test the functionality of the contract and its deployment.

USING BROWNIE TO INTERACT WITH SMART CONTRACTS

In the previous section of this tutorial, we learned how to deploy our simple storage smart contract. In this one, we are going to interact with it, meaning that we want to store a value inside the contract and retrieve it using Brownie. After doing this you will see how simply and efficiently Using Brownie to interact with smart contracts can get compared to other methods in python.

RETRIEVING THE STORED DATA:

So, to follow our deploy.py script, we have:

```
from Brownie import accounts, config, SimpleStorage
def deploy_simple_storage():

    account = accounts[0]
    simple_storage = SimpleStorage.deploy({"from": account})
    stored_value = simple_storage.retrieve()
    print(stored_value)
    transaction = simple_storage.store(38,{"from": account})
    transaction.wait(1)
    updated_stored_value = simple_storage.retrieve()
    print(updated_stored_value)

def main():
    deploy_simple_storage()
```

In the above code, we first try to retrieve the stored value inside the contract. Then, we store a number in it and again retrieve the stored value. By running this code using:

```
brownie run scripts/deploy.py
```

We have:

```
Brownie v1.18.1 - Python development framework for Ethereum
BrownieSimpleStorageProject is the active project. Launching
'ganache-cli --chain.vmErrorsOnRPCResponse true --server.port 8545
--miner.blockGasLimit 12000000 --wallet.totalAccounts 10 --hardfork
istanbul --wallet.mnemonic brownie'... Running
'scripts/deploy.py::main'... Transaction sent:
0xbbd21a1abc42f0d21f4651b71cddadddecf6ba99af3a31a140434950c7e36876
Gas price: 0.0 gwei Gas limit: 12000000 Nonce: 0
SimpleStorage.constructor confirmed Block: 1 Gas used: 334180
(2.78%) SimpleStorage deployed at:
0x3194cBDC3dbcd3E11a07892e7bA5c3394048Cc87 0 Transaction sent:
0x6bd512176120555da2d2682b4dd0256ccb8060a5c63d678d65ff4b554ffd7477
Gas price: 0.0 gwei Gas limit: 12000000 Nonce: 1 SimpleStorage.store
confirmed Block: 2 Gas used: 41393 (0.34%) SimpleStorage.store
confirmed Block: 2 Gas used: 41393 (0.34%) 38 Terminating local RPC
client…
```

As you can see, 2 blocks have been hashed at the end. And at first, we have nothing stored. So, the first number that has been retrieved is 0, and then after storing the number 38, we have it as the result of the retrieve.

TESTING SMART CONTRACTS USING BROWNIE

One of the important steps in developing a smart contract is to test our contract. Because in the main deployment of the transaction in a real-world smart contract if anything goes wrong, like spending infinitely extra ETH or mistakes like this, there will be no way to compensate for it. The tests folder is located in the main directory and is created by Brownie. We create a file named test_simple_storage.py and write:

```
from Brownie import SimpleStorage, accounts
def test_deploy():
    account = accounts[0]
    simple_storage = SimpleStorage.deploy({"from": account})
    starting_value = simple_storage.retrieve()
    expected = 0
    assert starting_value == expected
```

The above script does the same as deploy.py with the difference that it tests whether the output is the expected value or not. This will help us to understand the existing bugs without spending ETH.

```
brownie test
```

Result:

```
Brownie v1.18.1 - Python development framework for Ethereum
============================================================ test session
starts =============================================================
platform linux -- Python 3.8.10, pytest-6.2.5, py-1.11.0, pluggy-
1.0.0 rootdir: /home/mohamad/Desktop/solidity/Solidity and web3
codes/brownie_simple_storage plugins: eth-brownie-1.18.1, web3-
5.27.0, xdist-1.34.0, forked-1.4.0, hypothesis-6.27.3 collected 1
item Launching 'ganache-cli --chain.vmErrorsOnRPCResponse true --
server.port 8545 --miner.blockGasLimit 12000000 --
wallet.totalAccounts 10 --hardfork istanbul --wallet.mnemonic
brownie'... tests/test_simple_storage.py . [100%]
============================================================ 1 passed in
2.17s ==============================================================
Terminating local RPC client...
```

```
def test_deploy():
    account = accounts[0]
    simple_storage = SimpleStorage.deploy({"from": account})
    starting_value = simple_storage.retrieve()
    expected = 38
    assert starting_value == expected
```

```
brownie test
```

```
Brownie v1.18.1 - Python development framework for Ethereum
============================================================ test session
starts ============================================================
platform linux -- Python 3.8.10, pytest-6.2.5, py-1.11.0, pluggy-
1.0.0 rootdir: /home/mohamad/Desktop/solidity/Solidity and web3
codes/brownie_simple_storage plugins: eth-brownie-1.18.1, web3-
5.27.0, xdist-1.34.0, forked-1.4.0, hypothesis-6.27.3 collected 1
item Launching 'ganache-cli --chain.vmErrorsOnRPCResponse true --
server.port 8545 --miner.blockGasLimit 12000000 --
wallet.totalAccounts 10 --hardfork istanbul --wallet.mnemonic
brownie'... tests/test_simple_storage.py F [100%]
================================================================ FAILURES
================================================================

_____________________________________________________________
test_deploy
_____________________________________________________________ def
test_deploy(): account = accounts[0] simple_storage =
SimpleStorage.deploy({"from": account}) starting_value =
simple_storage.retrieve() expected = 38 > assert starting_value ==
expected E assert 0 == 38 tests/test_simple_storage.py:10:
AssertionError ============================================================
short test summary info
============================================================ FAILED
tests/test_simple_storage.py::test_deploy - assert 0 == 38
============================================================ 1 failed in
1.83s ============================================================
Terminating local RPC client… *****
```

As you can see, it shows failure because we haven't stored any value yet and the expected number should be 0 instead of 38. Now, let's define another function for storing the expected number, retrieving and putting it into the test:

```
from Brownie import SimpleStorage, accounts
def test_deploy():

    account = accounts[0]
    simple_storage = SimpleStorage.deploy({"from": account})
    starting_value = simple_storage.retrieve()
    expected = 38
    assert starting_value == expected

def test_updating_storage():

    account = accounts[0]
    simple_storage = SimpleStorage.deploy({"from": account})
    expected = 38
    simple_storage.store(expected,{"from":account})
    assert expected == simple_storage.retrieve()
```

```
brownie test
```

Result:

```
Brownie v1.18.1 - Python development framework for Ethereum
============================================= test session
starts ==============================================
platform linux -- Python 3.8.10, pytest-6.2.5, py-1.11.0, pluggy-
1.0.0 rootdir: /home/mohamad/Desktop/solidity/Solidity and web3
codes/brownie_simple_storage plugins: eth-brownie-1.18.1, web3-
5.27.0, xdist-1.34.0, forked-1.4.0, hypothesis-6.27.3 collected 2
items Launching 'ganache-cli --chain.vmErrorsOnRPCResponse true --
server.port 8545 --miner.blockGasLimit 12000000 --
wallet.totalAccounts 10 --hardfork istanbul --wallet.mnemonic
brownie'... tests/test_simple_storage.py F. [100%]
================================================== FAILURES
==================================================

_____________________________________________________
test_deploy
_____________________________________________________ def
test_deploy(): account = accounts[0] simple_storage =
SimpleStorage.deploy({"from": account}) starting_value =
simple_storage.retrieve() expected = 38 > assert starting_value ==
expected E assert 0 == 38 tests/test_simple_storage.py:10:
AssertionError ==========================================================
short test summary info
========================================== FAILED
tests/test_simple_storage.py::test_deploy - assert 0 == 38
======================================== 1 failed, 1 passed
in 2.07s ========================================
Terminating local RPC client…
```

As you can see, one test failed and the other passed. The expected result was not 38 at first but, after storing it as expected, it passed.

LAST THOUGHT ON USING BROWNIE TO INTERACT WITH SMART CONTRACTS

In this article, we have managed to retrieve the data we stored using the deploy.py python script. We have also created test.py files in the test directory to be able to see if the smart contract and the deploy.py work correctly. The Brownie module comes with a test package called pytest that makes the developers capable of testing any part of their script using the Brownie test command.