**Testing with Brownie**

[Brownie](https://github.com/iamdefinitelyahuman/brownie) is a Python-based development and testing framework for smart contracts. It includes a pytest plugin with fixtures that simplify testing your contract.

This section provides a quick overview of testing with Brownie. To learn more, you can view the Brownie documentation on [writing unit tests](https://eth-brownie.readthedocs.io/en/stable/tests-pytest-intro.html) or join the [Ethereum Python Dev Discord](https://discord.gg/abJEARdx3Q) #brownie channel.

**Getting Started**

In order to use Brownie for testing you must first [initialize a new project](https://eth-brownie.readthedocs.io/en/stable/init.html). Create a new directory for the project, and from within that directory type:

$ brownie init

This will create an empty [project structure](https://eth-brownie.readthedocs.io/en/stable/structure.html#structure) within the directory. Store your contract sources within the project’s contracts/ directory and your tests within tests/.

**Writing a Basic Test**

Assume the following simple contract Storage.vy. It has a single integer variable and a function to set that value.

1storedData: public(int128)

2

3**@external**

4**def** \_\_init\_\_(\_x: int128):

5 self.storedData = \_x

6

7**@external**

8**def** set(\_x: int128):

9 self.storedData = \_x

We create a test file tests/test\_storage.py where we write our tests in pytest style.

1**import** **pytest**

2

3INITIAL\_VALUE = 4

4

5

6**@pytest**.fixture

7**def** storage\_contract(Storage, accounts):

8 *# deploy the contract with the initial value as a constructor argument*

9 **yield** Storage.deploy(INITIAL\_VALUE, {'from': accounts[0]})

10

11

12**def** test\_initial\_state(storage\_contract):

13 *# Check if the constructor of the contract is set up properly*

14 **assert** storage\_contract.storedData() == INITIAL\_VALUE

15

16

17**def** test\_set(storage\_contract, accounts):

18 *# set the value to 10*

19 storage\_contract.set(10, {'from': accounts[0]})

20 **assert** storage\_contract.storedData() == 10 *# Directly access storedData*

21

22 *# set the value to -5*

23 storage\_contract.set(-5, {'from': accounts[0]})

24 **assert** storage\_contract.storedData() == -5

In this example we are using two fixtures which are provided by Brownie:

* accounts provides access to the [**Accounts**](https://eth-brownie.readthedocs.io/en/stable/api-network.html#brownie.network.account.Accounts) container, containing all of your local accounts
* Storage is a dynamically named fixture that provides access to a **[ContractContainer](https://eth-brownie.readthedocs.io/en/stable/api-network.html" \l "brownie.network.contract.ContractContainer" \o "(in Brownie v1.19.3))** object, used to deploy your contract

**Note**

To run the tests, use the brownie test command from the root directory of your project.

**Testing Events**

For the remaining examples, we expand our simple storage contract to include an event and two conditions for a failed transaction: AdvancedStorage.vy

1event DataChange:

2 setter: indexed(address)

3 value: int128

4

5storedData: public(int128)

6

7**@external**

8**def** \_\_init\_\_(\_x: int128):

9 self.storedData = \_x

10

11**@external**

12**def** set(\_x: int128):

13 **assert** \_x >= 0, "No negative values"

14 **assert** self.storedData < 100, "Storage is locked when 100 or more is stored"

15 self.storedData = \_x

16 log DataChange(msg.sender, \_x)

17

18**@external**

19**def** reset():

20 self.storedData = 0

To test events, we examine the **[TransactionReceipt](https://eth-brownie.readthedocs.io/en/stable/api-network.html" \l "brownie.network.transaction.TransactionReceipt" \o "(in Brownie v1.19.3))** object which is returned after each successful transaction. It contains an [**events**](https://eth-brownie.readthedocs.io/en/stable/api-network.html#brownie.network.event.EventDict) member with information about events that fired.

1**import** **brownie**

2

3INITIAL\_VALUE = 4

4

5

6**@pytest**.fixture

7**def** adv\_storage\_contract(AdvancedStorage, accounts):

8 **yield** AdvancedStorage.deploy(INITIAL\_VALUE, {'from': accounts[0]})

9

10**def** test\_events(adv\_storage\_contract, accounts):

11 tx1 = adv\_storage\_contract.set(10, {'from': accounts[0]})

12 tx2 = adv\_storage\_contract.set(20, {'from': accounts[1]})

13 tx3 = adv\_storage\_contract.reset({'from': accounts[0]})

14

15 *# Check log contents*

16 **assert** len(tx1.events) == 1

17 **assert** tx1.events[0]['value'] == 10

18

19 **assert** len(tx2.events) == 1

20 **assert** tx2.events[0]['setter'] == accounts[1]

21

22 **assert** **not** tx3.events *# tx3 does not generate a log*

**Handling Reverted Transactions**

Transactions that revert raise a **[VirtualMachineError](https://eth-brownie.readthedocs.io/en/stable/api-brownie.html" \l "brownie.exceptions.VirtualMachineError" \o "(in Brownie v1.19.3))** exception. To write assertions around this you can use **[brownie.reverts](https://eth-brownie.readthedocs.io/en/stable/api-test.html" \l "brownie.test.plugin.RevertContextManager" \o "(in Brownie v1.19.3))** as a context manager. It functions very similarly to **[pytest.raises](https://docs.pytest.org/en/latest/reference/reference.html" \l "pytest.raises" \o "(in pytest v7.4.0.dev88+gfbfd4b500))**.

[**brownie.reverts**](https://eth-brownie.readthedocs.io/en/stable/api-test.html#brownie.test.plugin.RevertContextManager) optionally accepts a string as an argument. If given, the error string returned by the transaction must match it in order for the test to pass.

1**import** **brownie**

2

3INITIAL\_VALUE = 4

4

5

6**@pytest**.fixture

7**def** adv\_storage\_contract(AdvancedStorage, accounts):

8 **yield** AdvancedStorage.deploy(INITIAL\_VALUE, {'from': accounts[0]})

9

10

11**def** test\_failed\_transactions(adv\_storage\_contract, accounts):

12 *# Try to set the storage to a negative amount*

13 **with** brownie.reverts("No negative values"):

14 adv\_storage\_contract.set(-10, {"from": accounts[1]})

15

16 *# Lock the contract by storing more than 100. Then try to change the value*

17

18 adv\_storage\_contract.set(150, {"from": accounts[1]})

19 **with** brownie.reverts("Storage is locked when 100 or more is stored"):

20 adv\_storage\_contract.set(10, {"from": accounts[1]})

21

22 *# Reset the contract and try to change the value*

23 adv\_storage\_contract.reset({"from": accounts[1]})

24 adv\_storage\_contract.set(10, {"from": accounts[1]})

25 **assert** adv\_storage\_contract.storedData() == 10