Smart Contract Developer Bootcamp Weekend Track: Day 2 Exercises: Brownie

Software Installation

Please ensure you've completed the setup Instructions before proceeding.

Exercise 1: My First Brownie Project

In this exercise, you'll use Brownie to create a new project that contains a simple smart contract that stores and retrieves a value, then you'll deploy it to the Kovan network and interact with it.

Setting Up Brownie

1. Open Visual Studio Code

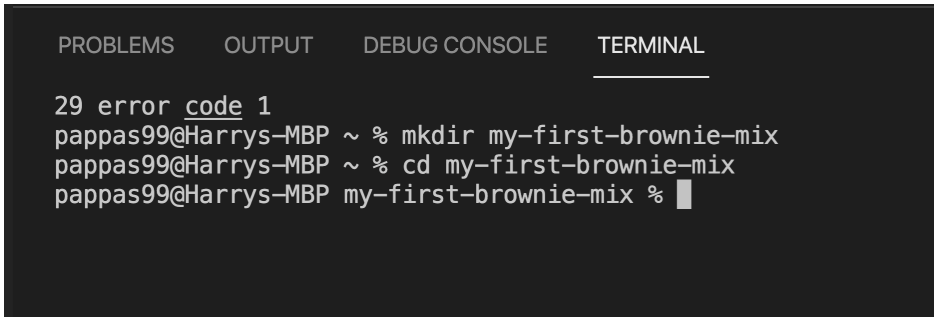
2. On the top header menu, choose View -> Terminal to bring up the VS Terminal (or press CTRL + `)

3. Create a new directory called 'my-first-brownie-mix' using the mkdir command.

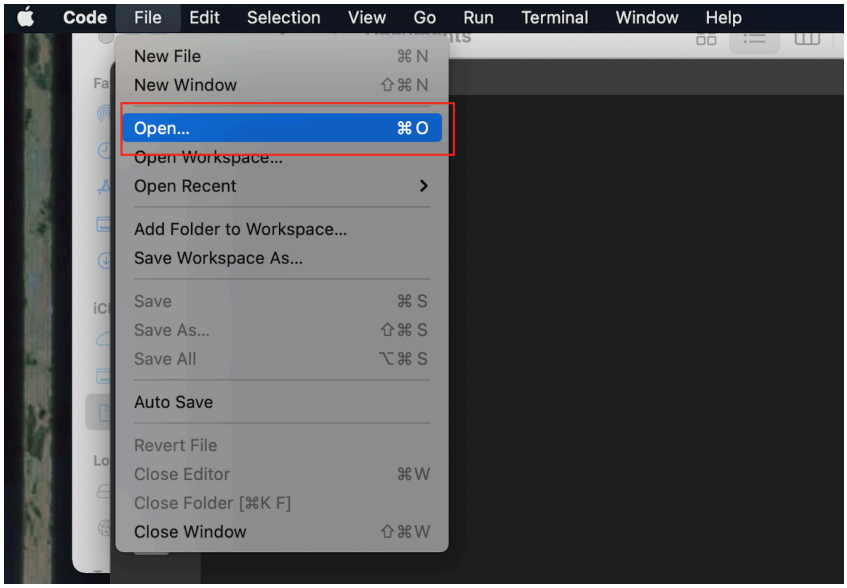
```
mkdir my-first-brownie-mix
```

4. Head into the directory by typing 'cd my-first-brownie-mix'

```
cd my-first-brownie-mix
```



5. In the top VS menu, choose File->Open(or Open Folder), then find your my-first-brownie-mix directory and choose Open. You should now see an explorer menu on the left hand side. If the terminal in VS Code is gone, re-open it by going to View->Terminal (or CTRL + `)



6. We're now ready to create a new Brownie project. Check to see if you have Brownie installed with the following command

```
brownie
```

You should get output similar to the following

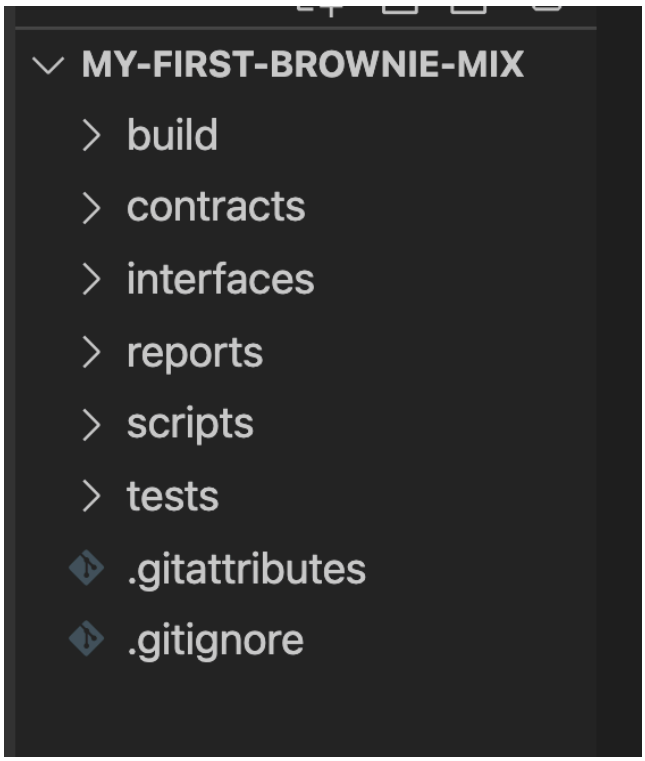
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Brownie v1.14.6 - Python development framework for Ethereum

Usage:  brownie <command> [<args>...] [options <args>]

Commands:
  init              Initialize a new brownie project
  bake              Initialize from a brownie-mix template
  pm                Install and manage external packages
  compile           Compile the contract source files
  console           Load the console
  test              Run test cases in the tests/ folder
  run               Run a script in the scripts/ folder
  accounts          Manage local accounts
  networks          Manage network settings
  gui               Load the GUI to view opcodes and test coverage
  analyze           Find security vulnerabilities using the MythX API

Options:
  --help -h         Display this message
  --version         Show version and exit

Type 'brownie <command> --help' for specific options and more information about
each command.
pappas99@Harrys-MBP my-first-brownie-mix %
```

If you get an error, please refer to the installing brownie section of the Brownie setup instructions web page, and follow the steps there. Alternatively, check out the Troubleshooting section of this exercises document for other hints.

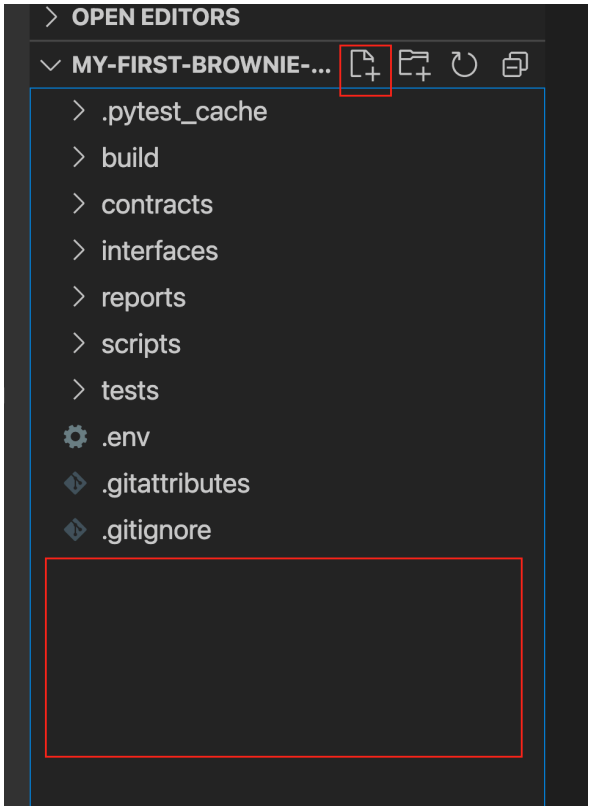
7. Now you're ready to create a new brownie project. First, initialize a new project with the following command in your VS Code terminal

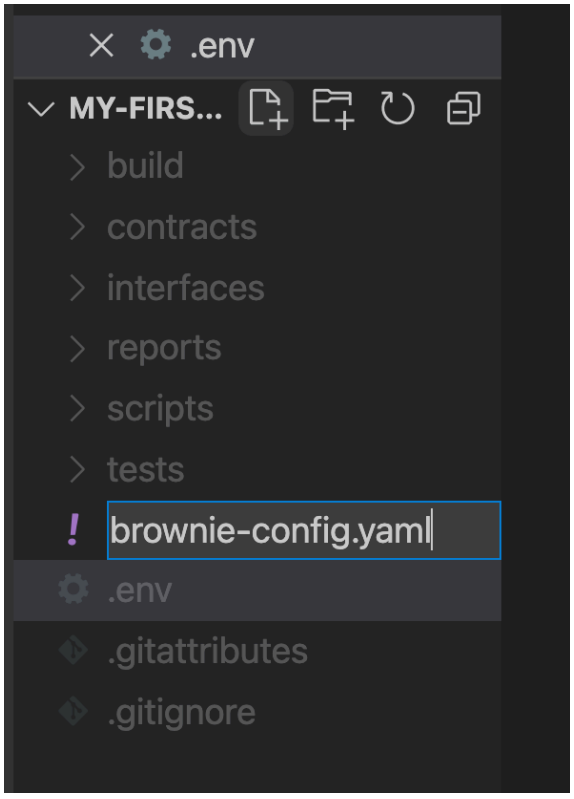
```
brownie init
```

Your folder structure for your project should now look like this



8. First thing you need to do is create a new config file. For Brownie, this file is called 'brownie-config.yaml', and it contains a number of configuration settings. We'll leave most of them as the brownie default values (ie, we don't need to specify them in the config file), but we'll add a couple custom config entries in. Create a new file in the 'my-first-brownie-mix' folder, called 'brownie-config.yaml'. To ensure you create the file in the top level folder, you can click the mouse in the blank space under the folder structure, it should highlight blue, and you can then click on the new file icon

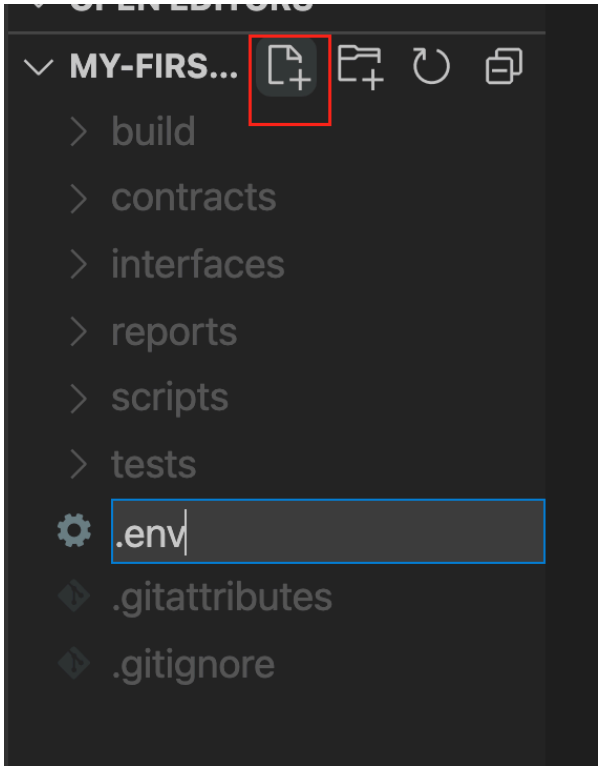

OPEN EDITORS
MY-FIRST-BROWNIE-...
.pytest_cache
build
contracts
interfaces
reports
scripts
tests
.env
.gitattributes
.gitignore


.env
MY-FIRS...
build
contracts
interfaces
reports
scripts
tests
brownie-config.yaml
.env
.gitattributes
.gitignore

9. Put the following entry into the config file, then save it. This config tells Brownie do the following:
    - To obtain sensitive environment variables from a .env file
    - References our private key from the .env file for doing deployments. This means you don't need to manually interact with MetaMask and sign transactions. You will let brownie handle it all for you automatically.

```
dotenv: .env
wallets:
  from_key: ${PRIVATE_KEY}
```

10. Next you need to set your environment variables. **WEB3_INFURA_PROJECT_ID** and **PRIVATE_KEY**. In the explorer, directly to the right of your folder name 'MY-FIRST-BROWNIE-MIX', press the new file button to create a new file, call it **.env.** Once again, ensure the top level folder is selected to ensure you create the file in the top folder.



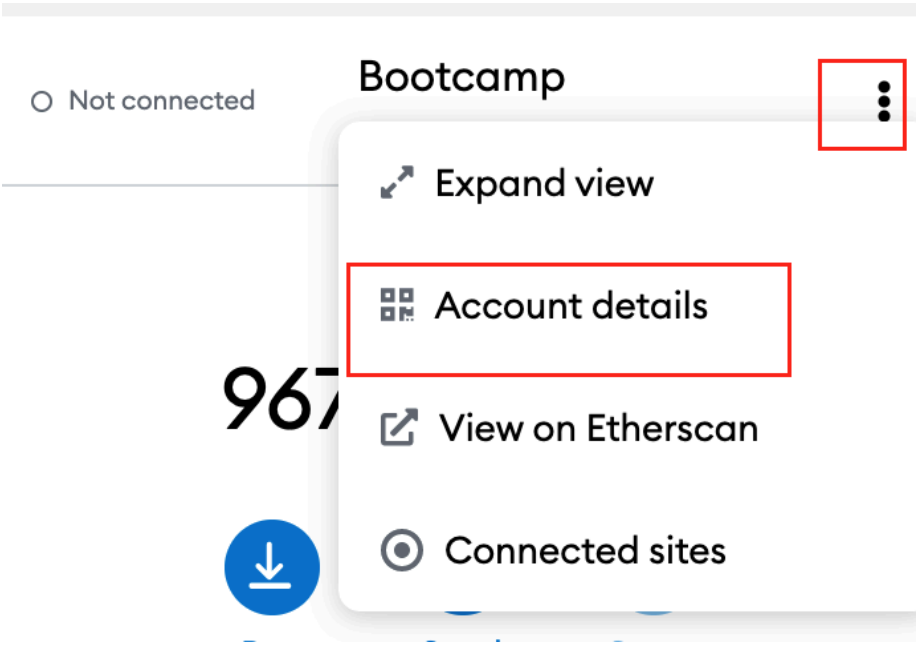
11. Open your newly created .env file in the editor by double clicking on it in the explorer window. Put the following text in the file, then save it.

```
### uncomment me to use the variables
export WEB3_INFURA_PROJECT_ID='aaa5aa5a5a5a55555aaa555a5a5555a'
export PRIVATE_KEY='0xasdfasdfasdfasdfasdfasdfasdfas'

### If you want to verify contracts on etherescan
# export ETHERSCAN_TOKEN='asdfadfasdfsf'
```

12. Replace the contents of the WEB3_INFURA_PROJECT_ID string with the ending string in the URL that you saved when you signed up for a free Infura key as part of the setup instructions. It should be something like this: **soadhjfjdks8400975984slhfdskjdhf498**

13. Now you need to put your MetaMask wallet account private key in the PRIVATE_KEY environment variable. Open up MetaMask, press on the three dots to the right of your account name, then select Account Details -> Export private key, enter in your password, then copy your private key string, and paste it into your .env file PRIVATE_KEY string. You'll need to add `0x` to the start of the private key.

NOTE: BE CAREFUL WITH COPYING AND PASTING PRIVATE KEYS FOR ACCOUNTS THAT HAVE MAINNET FUNDS IN THEM



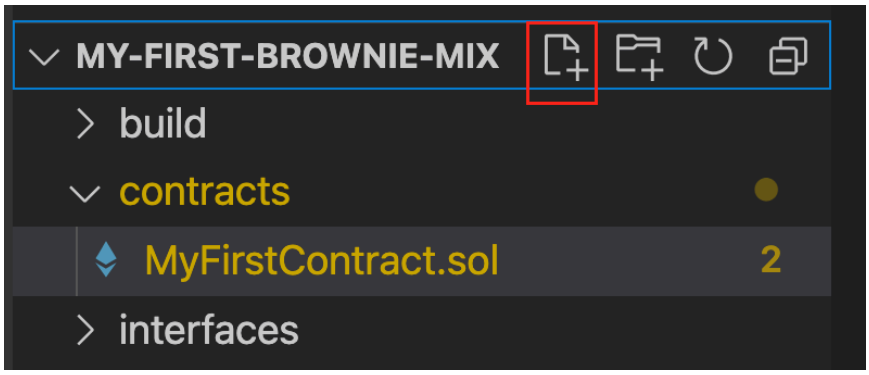
14. Your .env file should look something like this (but with different hash values). Save the file.

```
### uncomment me to use the variables
export WEB3_INFURA_PROJECT_ID='37asdf43sc44eb6asdf4e5be504c4979f'
export
PRIVATE_KEY='0f77asdfsdkllfkh4389543lk5h4lk35h43y5h34jh5jk43h5k4j3k3j45h1
fe210d'

### If you want to verify contracts on etherescan
# export ETHERSCAN_TOKEN='asdfadfasdfsf'
```

You've now set up your brownie config and .env file. Now you're ready to create the smart contract!

Creating The Smart Contract

1. Select the contract folder in the explorer, then select the ‘New File’ icon, and create a new file, call it ‘MyFirstContract.sol’, and press enter.

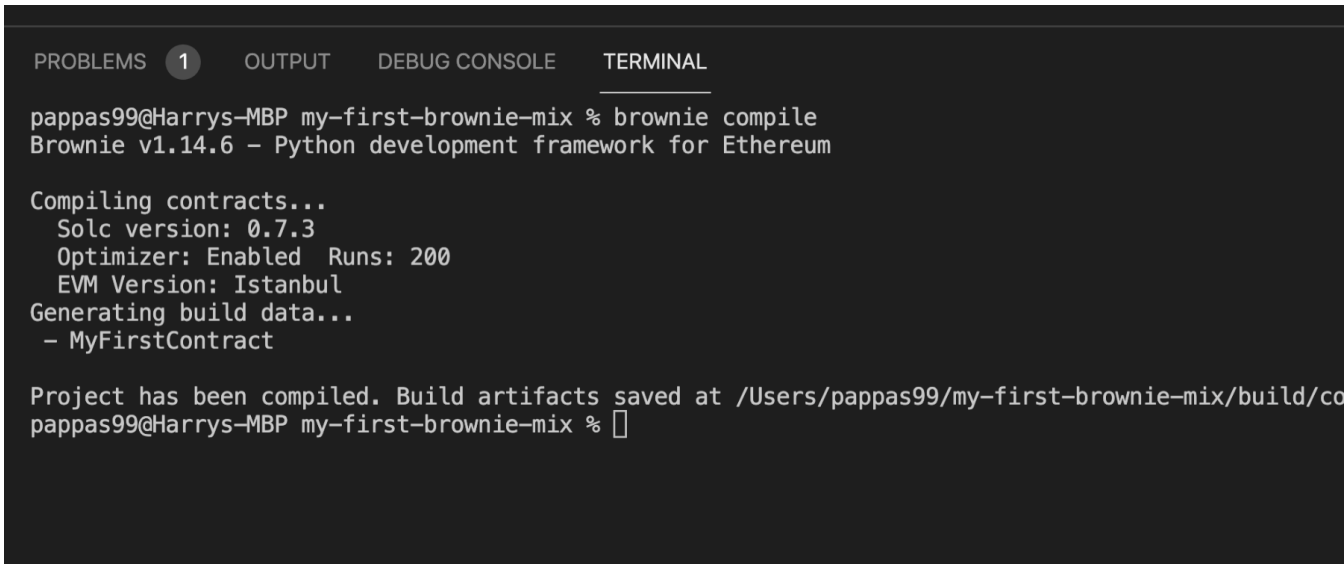


2. If it isn’t already open, open your new MyFirstContract.sol file by double clicking it in the explorer menu. Paste the following into the smart contract file, then save your changes. This is the same smart contract from yesterday's Remix exercises.

```
pragma solidity =0.7.3;

contract MyFirstContract {

    uint256 number;


    function setNumber(uint256 _num) public {
        number = _num;
    }


    function getNumber() public view returns (uint256){
        return number;
    }
}
```

3. Now that you've created your smart contract, you can compile it. In the terminal, type in the following command to compile your smart contract

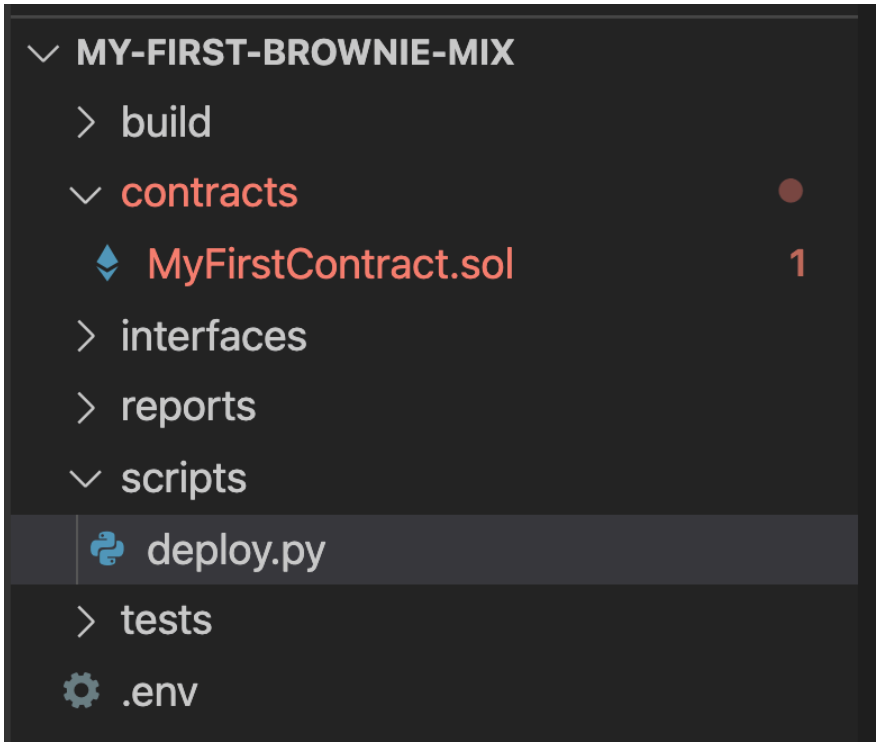
```
brownie compile
```



4. You now have our smart contract ready to go! Next you'll create a script to deploy the smart contract to the Kovan network, and execute the two functions in it

Deploying and Interacting With the Smart Contract

5. Inside the scripts folder, create a new file called 'deploy.py'.



6. Paste the following code into the newly created file, then save your changes. This script will simply deploy the contract to the specified network, using your account/wallet details specified in the brownie-config.yaml file.

```
from brownie import MyFirstContract, config, accounts

def deployContract():
    account = accounts.add(config["wallets"]["from_key"]) or
accounts[0]
    MyFirstContract.deploy({'from': account})

def main():
    deployContract()
```

7. We're now ready to deploy our smart contract to the Kovan network or a local network. Back in the terminal, enter the following command to deploy your smart contract to the Kovan network. If you get an error stating you have insufficient funds, ensure your wallet in MetaMask has some ETH in it, as per the [setup instructions.]

```
brownie run deploy.py --network kovan
```

8. You should see your contract deployed to a new contract address on the Kovan network

```
pappas99@Harrys-MBP my-first-brownie-mix % brownie run deploy.py --network kovan
Brownie v1.14.6 - Python development framework for Ethereum

Compiling contracts...
  Solc version: 0.7.3
  Optimizer: Enabled  Runs: 200
  EVM Version: Istanbul
Generating build data...
 - MyFirstContract

MyFirstBrownieMixProject is the active project.

Running 'scripts/deploy.py::main'...
Transaction sent: 0xde36fe91c4955bc26c85b3f0b5014693d45acdabd54515230d6ed4c716d2f661
  Gas price: 5.0 gwei   Gas limit: 99619   Nonce: 669
  MyFirstContract.constructor confirmed - Block: 25732346   Gas used: 90563 (90.91%)
  MyFirstContract deployed at: 0xddBbEf30bfd85EE8EAe0e7BA34517aB893F1C1b7

pappas99@Harrys-MBP my-first-brownie-mix %
```

9. Next you'll create a script to interact with the deployed project. In the 'scripts' folder once again, create a new file called 'interact.py', and put the following into the file and save your work. This file will grab your deployed contract, and call the 'setNumber' and 'getNumber' functions in it.

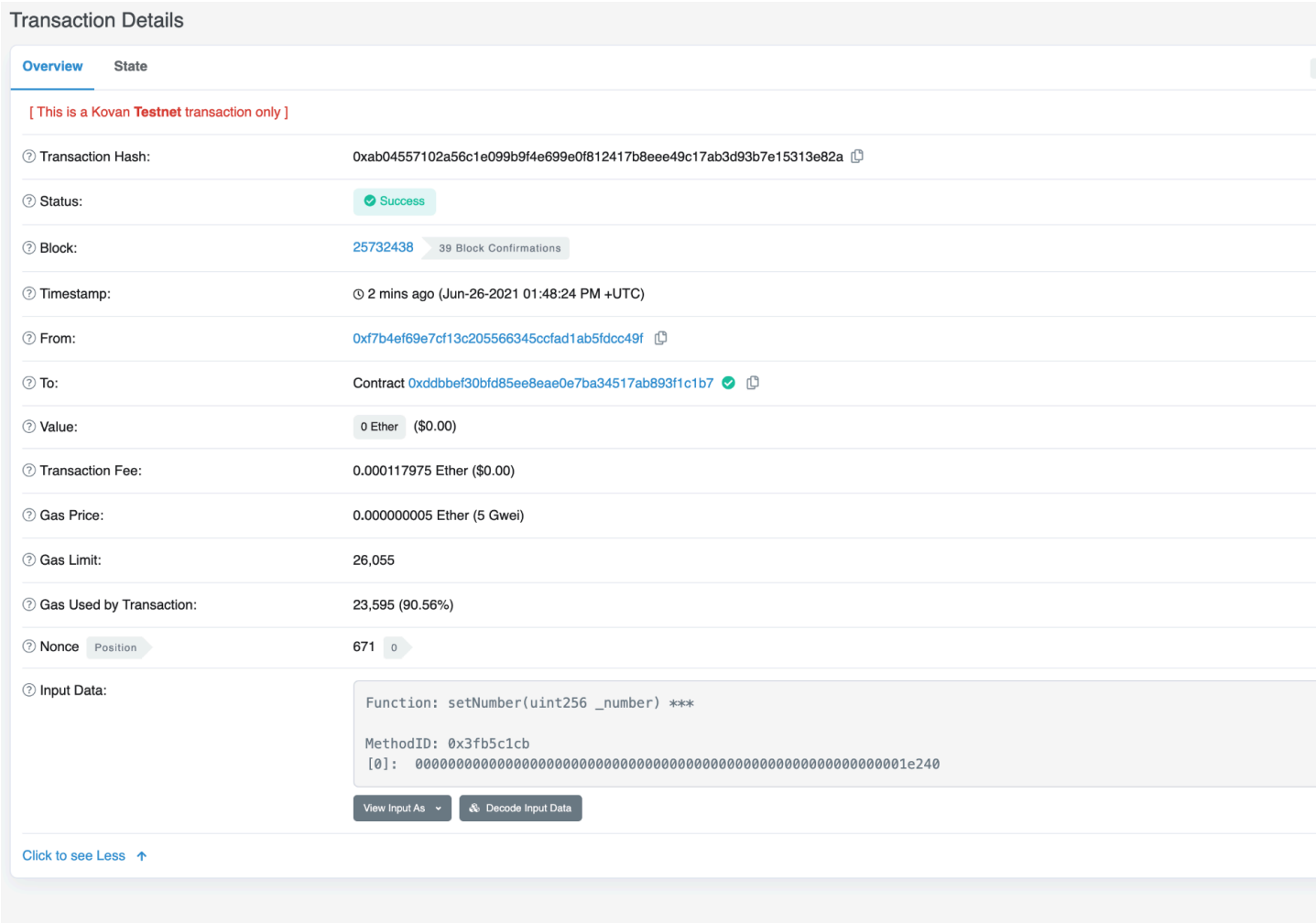
```
from brownie import MyFirstContract, config, accounts, network

def main():
    account = accounts.add(config["wallets"]["from_key"])
    myFirstContract = MyFirstContract[-1]
    tx = myFirstContract.setNumber(123456,{'from': account})
    tx.wait(1)
    print("Number is", myFirstContract.getNumber())
```

10. Run your newly created script by entering the following command into the Terminal.

```
brownie run interact.py --network kovan
```

```
pappas99@Harrys-MBP my-first-brownie-mix % brownie run interact.py --network kovan
Brownie v1.14.6 - Python development framework for Ethereum

Compiling contracts...
  Solc version: 0.7.3
  Optimizer: Enabled  Runs: 200
  EVM Version: Istanbul
Generating build data...
 - MyFirstContract

MyFirstBrownieMixProject is the active project.

Running 'scripts/interact.py::main'...
Transaction sent: 0xab04557102a56c1e099b9f4e699e0f812417b8eee49c17ab3d93b7e15313e82a
  Gas price: 5.0 gwei   Gas limit: 26055   Nonce: 671
  MyFirstContract.setNumber confirmed - Block: 25732438   Gas used: 23595 (90.56%)

This transaction already has 2 confirmations.
Number is 123456
pappas99@Harrys-MBP my-first-brownie-mix %
```

11. Copy the transaction hash specified in the output (next to ‘Transaction Sent’, and search for it on https://kovan.etherscan.io/. You should see the details of the transaction



Congratulations, you've successfully created a new smart contract project using the Brownie Smart Contract Platform, deployed it to a public test network, and interacted with your deployed smart contract!

Bonus Exercises:

You can attempt to complete these exercises if you've completed the main exercise ahead of schedule:

1. Add a view function to your contract that takes a uint input parameter, and returns the sum of the stored number and the input number. The function return type should be uint. Ensure your new contract compiles

2. Modify the interact script in your project to also interact with the new function

3. Create a new contract in the contracts folder called ERC-20.sol. Open the file, and enter in the full source code for the ERC20 token contract example on the Ethereum developer tutorials page. The only changes you need to make is to update the pragma Solidity version at the top of the contract, and to comment out the Approval and Transfer events in the ERC20Basic contract to avoid compilation errors. You can also update the symbol variable from ERC to whatever you wish. Save your file.

```
pragma solidity =0.7.3;

interface IERC20 {

    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function allowance(address owner, address spender) external view
returns (uint256);

    function transfer(address recipient, uint256 amount) external returns
(bool);
    function approve(address spender, uint256 amount) external returns
(bool);
    function transferFrom(address sender, address recipient, uint256
amount) external returns (bool);


    event Transfer(address indexed from, address indexed to, uint256
value);
    event Approval(address indexed owner, address indexed spender,
uint256 value);
}


contract ERC20Basic is IERC20 {

    string public constant name = "ERC20Basic";
    string public constant symbol = "ERC";
    uint8 public constant decimals = 18;


    //event Approval(address indexed tokenOwner, address indexed spender,
uint tokens);
```

```
    //event Transfer(address indexed from, address indexed to, uint
tokens);


    mapping(address => uint256) balances;

    mapping(address => mapping (address => uint256)) allowed;

    uint256 totalSupply_;

    using SafeMath for uint256;


   constructor(uint256 total) public {
    totalSupply_ = total;
    balances[msg.sender] = totalSupply_;
    }

    function totalSupply() public override view returns (uint256) {
    return totalSupply_;
    }

    function balanceOf(address tokenOwner) public override view returns
(uint256) {
        return balances[tokenOwner];
    }

    function transfer(address receiver, uint256 numTokens) public
override returns (bool) {
        require(numTokens <= balances[msg.sender]);
        balances[msg.sender] = balances[msg.sender].sub(numTokens);
        balances[receiver] = balances[receiver].add(numTokens);
        emit Transfer(msg.sender, receiver, numTokens);
        return true;
    }

    function approve(address delegate, uint256 numTokens) public override
returns (bool) {
        allowed[msg.sender][delegate] = numTokens;
        emit Approval(msg.sender, delegate, numTokens);
        return true;
```

```
    }

    function allowance(address owner, address delegate) public override
view returns (uint) {
        return allowed[owner][delegate];
    }

    function transferFrom(address owner, address buyer, uint256
numTokens) public override returns (bool) {
        require(numTokens <= balances[owner]);
        require(numTokens <= allowed[owner][msg.sender]);

        balances[owner] = balances[owner].sub(numTokens);
        allowed[owner][msg.sender] =
allowed[owner][msg.sender].sub(numTokens);
        balances[buyer] = balances[buyer].add(numTokens);
        emit Transfer(owner, buyer, numTokens);
        return true;
    }
}

library SafeMath {
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
      assert(b <= a);
      return a - b;
    }

    function add(uint256 a, uint256 b) internal pure returns (uint256) {
      uint256 c = a + b;
      assert(c >= a);
      return c;
    }
}
```

4. Create a new script in the scripts folder called ‘deploy-erc20.py’. Base the contents of the deploy script on your existing ‘deploy.py’ file.

5. Modify the deploy script so that it deploys your ‘ERC20Basic’ contract instead of the ‘MyFirstContract’ one. You should do the following:

a. Change the import to ERC20Basic
b. Change any variables related to MyFirstSmartContract to ERC20Basic
c. Specify an initial token supply in the constructor in the .deploy command. It should be the first value, then add a ‘,’ before the {‘from: account} part

Spoiler/Solution (highlight to see)

```
```

6. Compile all contracts in your project, then execute your new script

```
brownie compile
```

```
brownie run deploy_erc20.py --network kovan
```

7. Look for your deployed contract on https://kovan.etherscan.io/.

8. Create a new script ‘interact_erc20.py’ to mint and send yourself some tokens with the deployed contract by calling the required functions, and passing in the parameters. For now you can hardcode your wallet address values, you can obtain them from MetaMask. I.e send tokens from your current account that you have setup in the private key environment variable, and send them to another wallet address in your MetaMask.

Exercise 2: Brownie Starter Kit

In this exercise, you’ll download the Brownie Starter Kit, and use it to create, deploy and execute smart contracts that use Chainlink Data Feeds, Any-API and VRF. This will get you more familiar with working with smart contracts and scripts in the brownie development environment.

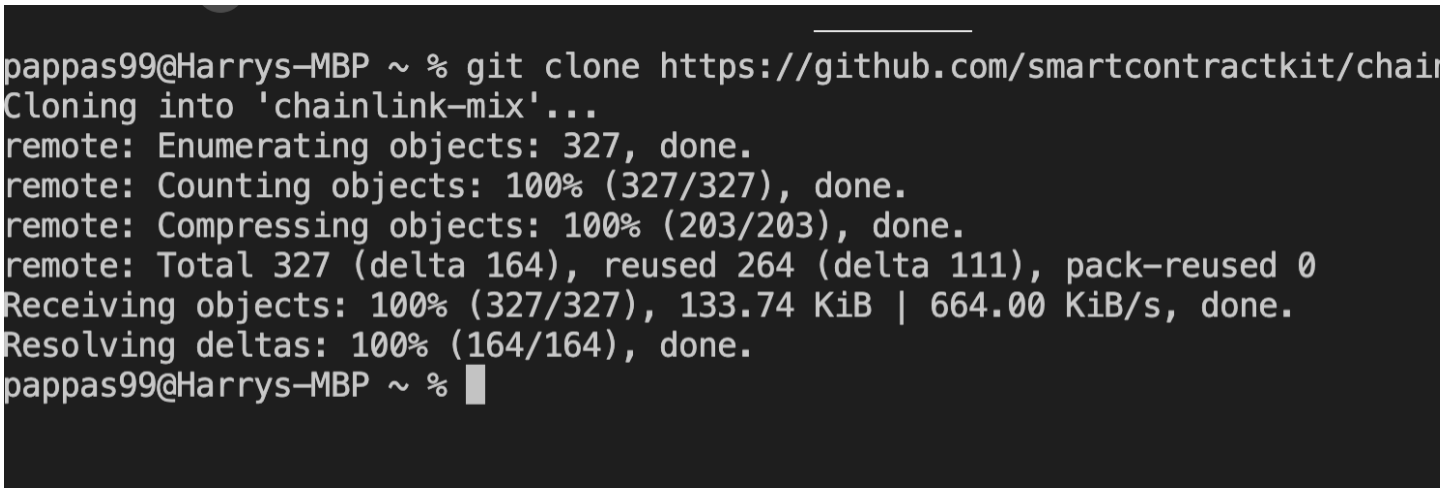
Downloading the Brownie Starter Kit

1. In VS Code, go to your terminal, then go back to your home folder by typing in ‘cd ..’ and pressing enter

```
cd ..
```

2. Pull a copy of the Brownie Starter Kit with the following command:

```
git clone https://github.com/smartcontractkit/chainlink-mix
```

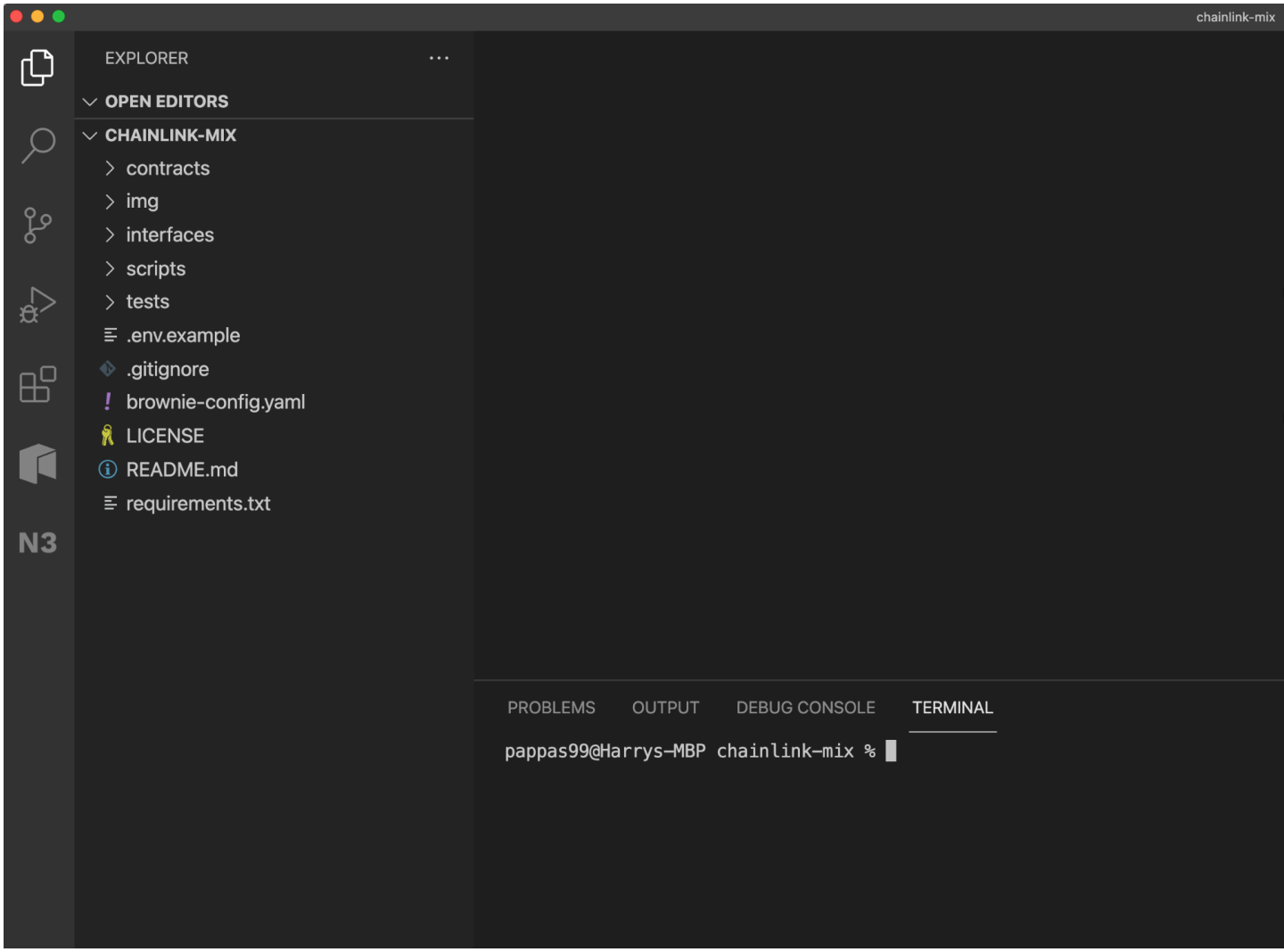


Optionally instead of git clone, you could run `brownie bake chainlink-mix` and then `cd chainlink`

3. In your terminal, you need to change the directory to your copy of the Brownie Starter Kit that you downloaded in the previous step.

```
cd chainlink-mix
```

4. Because we now have a new project in a new folder, we need to switch to it in Visual Studio Code. Choose File-> Open, find the ‘chainlink-mix’ folder, select it and press Open. If your VS Terminal has disappeared, re-open it by selecting View-> Terminal (or pressing CTRL + `). Your VS Code should look like this:



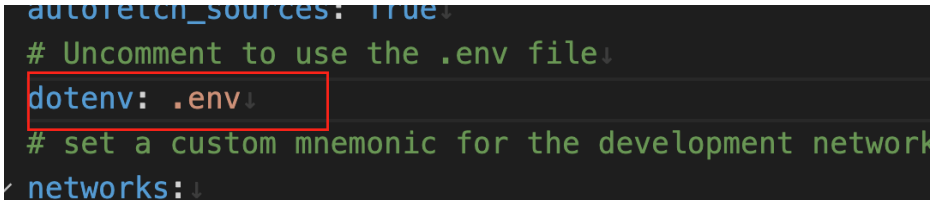
Setting Up The Brownie Starter Kit

1. First thing you'll need to do is once again create a .env file in the project folder, and set both WEB3_INFURA_PROJECT_ID and PRIVATE_KEY values, exactly like you did in the previous exercise in the '[Setting Up Brownie]' section. Your .env file should look something like this (with different hash values). If you have your .env file from your previous 'My First Brownie Project' project, you can also copy that file into this project folder instead of recreating one.

```
### uncomment me to use the variables
export WEB3_INFURA_PROJECT_ID='37asdf43sc44eb6asdf4e5be504c4979f'
export
PRIVATE_KEY='0f77asdfsdkllfkh4389543lk5h4lk35h43y5h34jh5jk43h5k4j3k3j45h1
fe210d'
```

```
### If you want to verify contracts on etherescan
# export ETHERSCAN_TOKEN='asdfadfasdfsf'
```

2. Open the ‘brownie-config.yaml’ file in the project file browser, and uncomment the line ‘dotenv: .env’ by removing the leading ‘#’, then save the file. This will tell Brownie to use your .env file for loading environment variables.

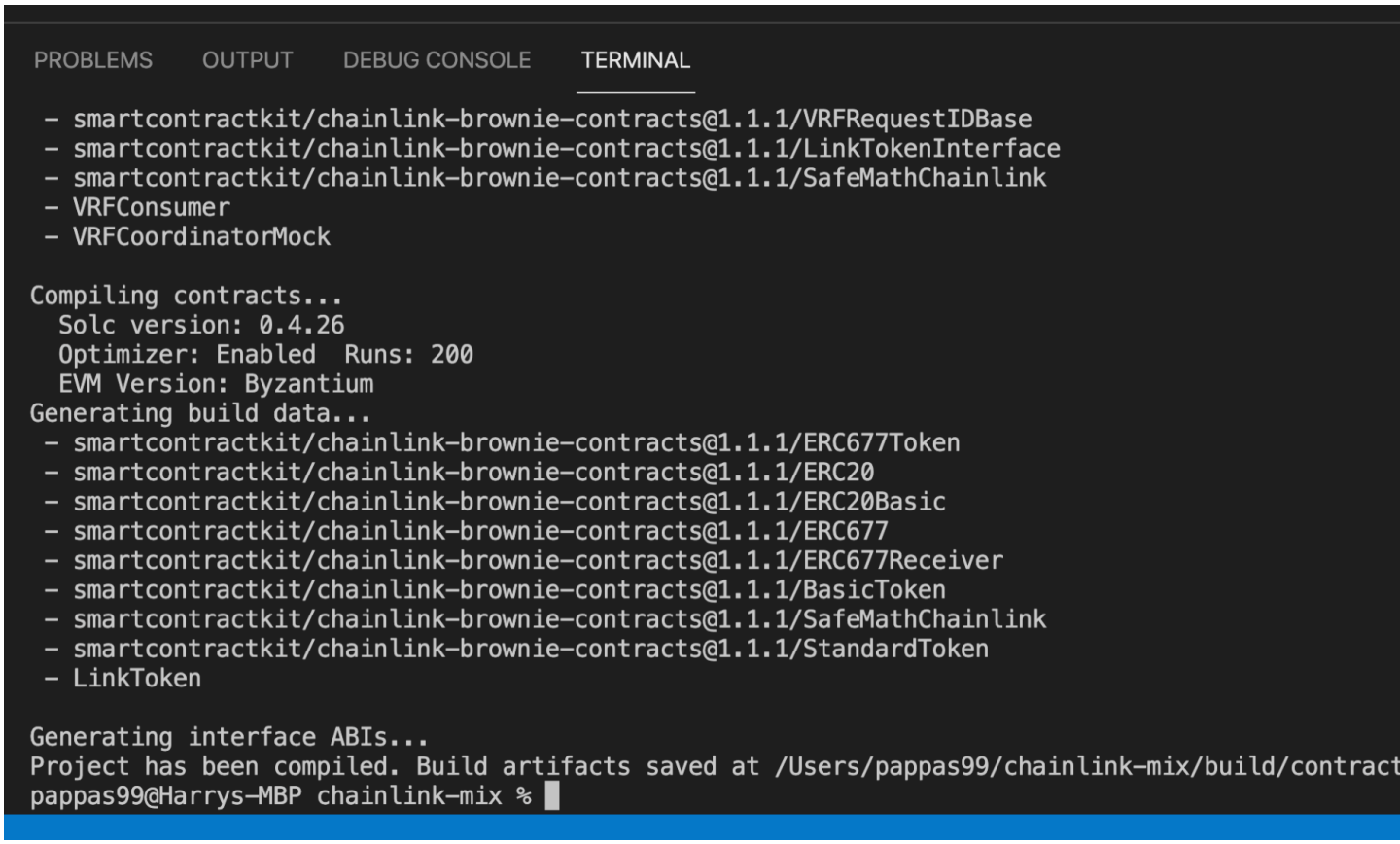


3. Your brownie starter kit configuration is now ready, and we’re ready to deploy our smart contracts

Compiling and Deploying The Smart Contracts

1. Before you deploy the smart contracts in the brownie starter kit to the Kovan test network, you should compile them first. Run the following command in the terminal

```
brownie compile
```



Price Feed Contract

The Price Feed Consumer contract makes use of Chainlink Price Feeds to fetch the current ETH/USD price.

2. To deploy the PriceFeedConsumer contract, run the following command in the terminal. It will execute a script that deploys the compiled contract to the Kovan test network

```
brownie run scripts/price_feed_scripts/01_deploy_price_consumer_v3.py
--network kovan
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

pappas99@Harrys-MBP chainlink-mix % brownie run scripts/price_feed_scripts/01_deploy_price
Brownie v1.14.6 - Python development framework for Ethereum

ChainlinkMixProject is the active project.

Running 'scripts/price_feed_scripts/01_deploy_price_consumer_v3.py::main'...
Transaction sent: 0x2bde275fd29fe962ec73c4461b4c4512e4c6f6d3f5b15f5629332b6157d72506
  Gas price: 5.0 gwei   Gas limit: 147030   Nonce: 672
  PriceFeedConsumer.constructor confirmed - Block: 25733147   Gas used: 133664 (90.91%)
  PriceFeedConsumer deployed at: 0xA458D87e96ee13dF3Ead31C463E8EDe2D2adc036

The current price of ETH is 175375509793
pappas99@Harrys-MBP chainlink-mix %
```

3. To interact with the deployed Price Feed Consumer contract, run the '02_read_price_feed.py' script, which will query the deployed smart contract, and return the latest price of the specified price feed.

```
brownie run scripts/price_feed_scripts/02_read_price_feed.py --network
kovan
```

```
pappas99@Harrys-MBP chainlink-mix % brownie run scripts/price_feed_scripts/02_read_pric
Brownie v1.14.6 - Python development framework for Ethereum

ChainlinkMixProject is the active project.

Running 'scripts/price_feed_scripts/02_read_price_feed.py::main'...
Reading data from 0xA458D87e96ee13dF3Ead31C463E8EDe2D2adc036
175375509793
pappas99@Harrys-MBP chainlink-mix %
```

You have now successfully consumed Chainlink Data Feeds in your smart contract created and deployed using the brownie development environment.

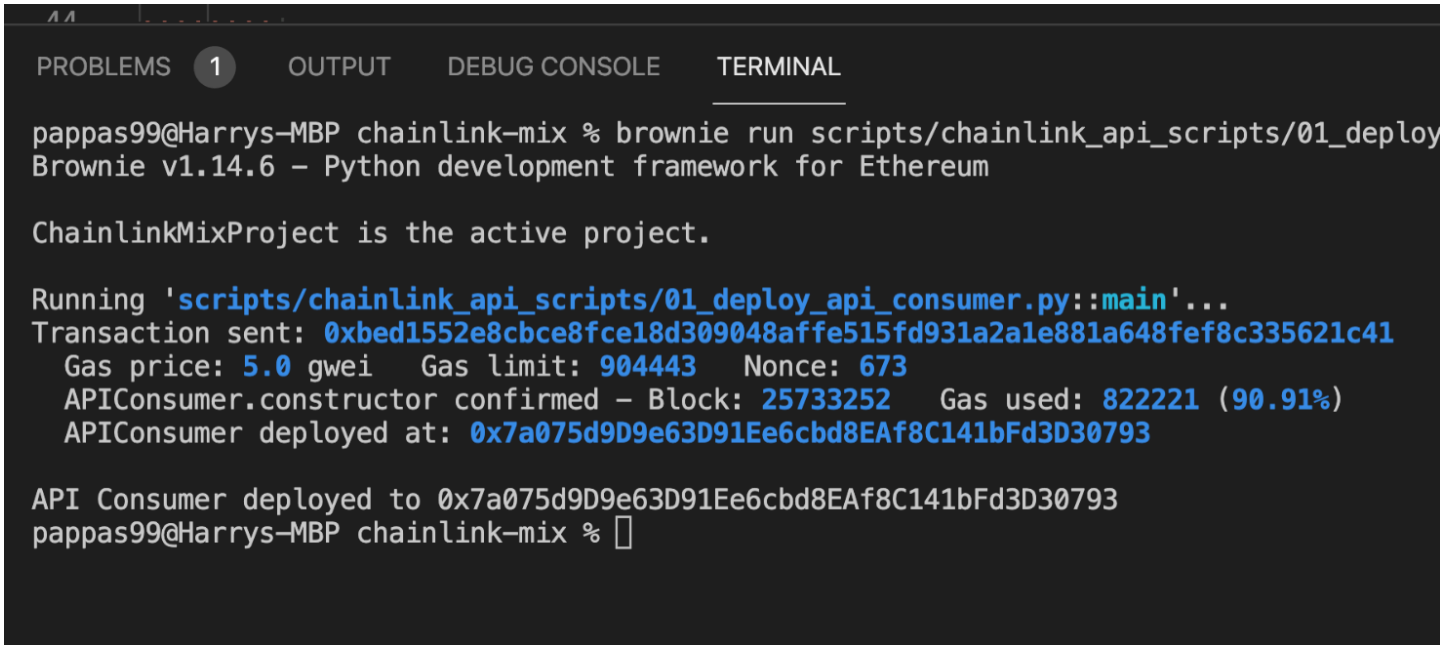
You can do the deployment script, without the --network kovan flag, and you'll run the script in a temporary ganache chain as well.

API Consumer Contract

The API Consumer Contract makes an API call via using Chainlink Any-API

4. To deploy the APIConsumer contract, run the following command in the terminal. It will execute a script that deploys the compiled contract to the Kovan test network

```
brownie run scripts/chainlink_api_scripts/01_deploy_api_consumer.py
--network kovan
```



5. To interact with the deployed API Consumer contract, run the '02_request_api.py' script. This script will fund the deployed contract with some LINK from your MetaMask wallet, and then it will execute the 'requestVolumeData' function in the contract, which will reach out to the cryptocompare API to get the current ETH/USD volume

```
brownie run scripts/chainlink_api_scripts/02_request_api.py --network
kovan
```

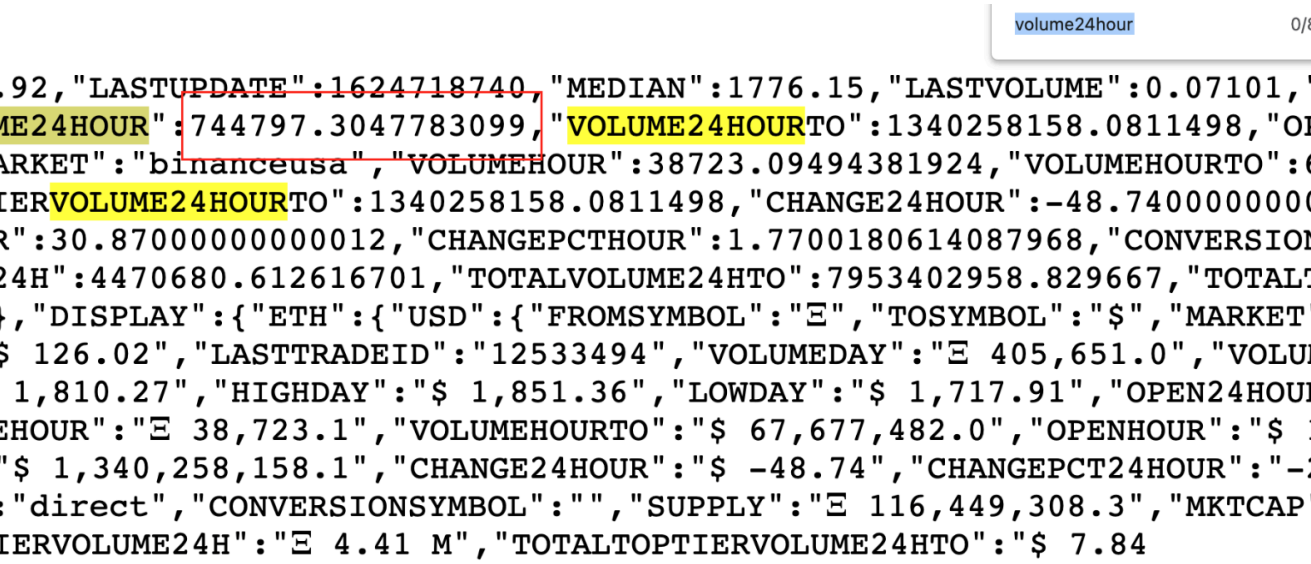
```
36      * data, then multiply by 1000000000000000000 (to remove decimal places from da

PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL

pappas99@Harrys-MBP chainlink-mix % brownie run scripts/chainlink_api_scripts/02_request_api.
Brownie v1.14.6 - Python development framework for Ethereum

ChainlinkMixProject is the active project.

Running 'scripts/chainlink_api_scripts/02_request_api.py::main'...
Transaction sent: 0x424bdac9a868c9301328ee728addc8f4070271ee8185acafc2325d2e796f73ab
  Gas price: 5.0 gwei   Gas limit: 57703   Nonce: 674
  LinkTokenInterface.transfer confirmed - Block: 25733264   Gas used: 52458 (90.91%)

Funded 0x7a075d9D9e63D91Ee6cbd8EAf8C141bFd3D30793
  LinkTokenInterface.transfer confirmed - Block: 25733264   Gas used: 52458 (90.91%)

Transaction sent: 0xcbc95ed68b4492f643d22a835fd453f34e18071a23bca4990dc56b0609e17360
  Gas price: 5.0 gwei   Gas limit: 147117   Nonce: 675
  APIConsumer.requestVolumeData confirmed - Block: 25733266   Gas used: 118743 (80.71%)

pappas99@Harrys-MBP chainlink-mix %
```

6. To see the result of the API call in your smart contract, run the '03_read_data.py' script. Ensure a few seconds has passed since you ran the previous '02_request_api' script, to ensure the Chainlink Oracle has successfully made the callback transaction

```
brownie run scripts/chainlink_api_scripts/03_read_data.py --network kovan
```

```
36      * data, then multiply by 1000000000000000000 (to remove decimal places fr

PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL

pappas99@Harrys-MBP chainlink-mix % brownie run scripts/chainlink_api_scripts/03_read_da
Brownie v1.14.6 - Python development framework for Ethereum

ChainlinkMixProject is the active project.

Running 'scripts/chainlink_api_scripts/03_read_data.py::main'...
Reading data from 0x7a075d9D9e63D91Ee6cbd8EAf8C141bFd3D30793
744074453447579900000000
pappas99@Harrys-MBP chainlink-mix %
```

You have now successfully performed a request for external data, and obtained the result into your smart contract using a Chainlink oracle. You can verify the result by opening a browser window, heading to https://min-api.cryptocompare.com/data/pricemultifull?fsyms=ETH&tsyms=USD and then searching for the string **"VOLUME24HOUR"**. As per the logic in the smart contract, the result has been multiplied by 10**18 to avoid any floating point numbers.

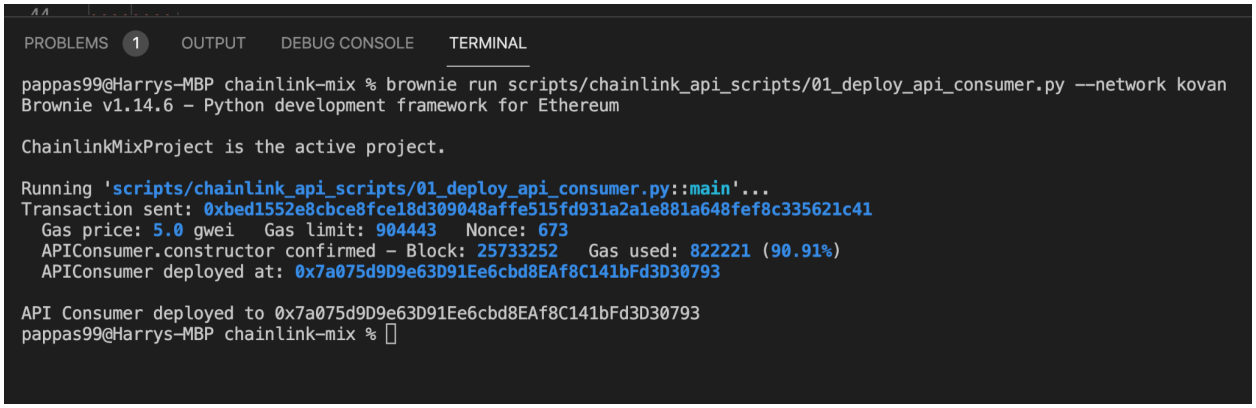


VRF Consumer Contract

The VRF Consumer contract performs a request for randomness using Chainlink VRF, and then stores the resulting random number.

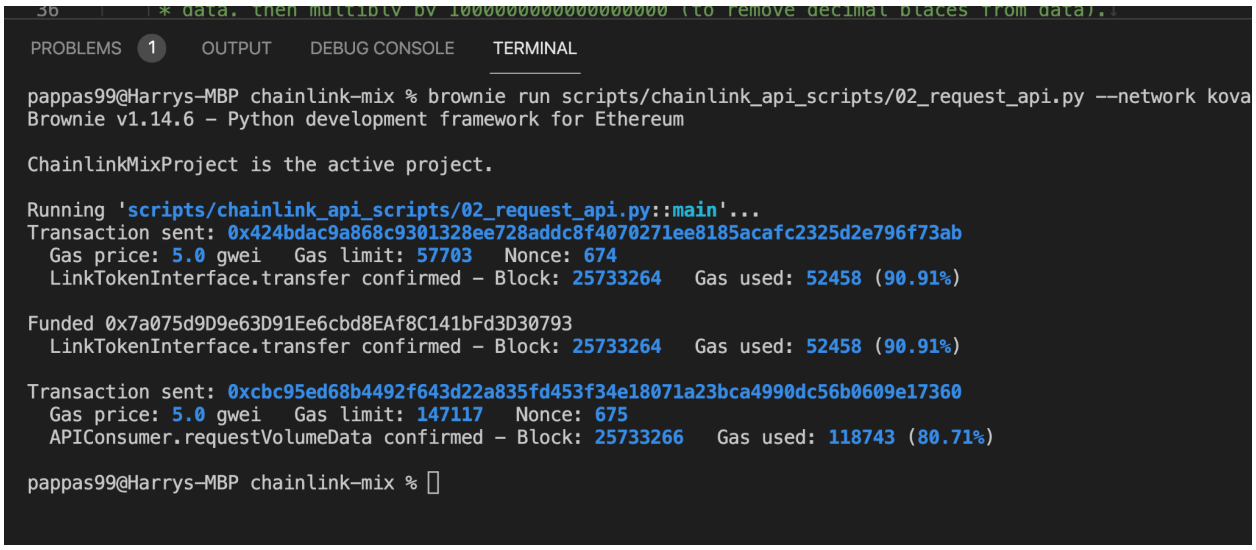
7. To deploy the VRFConsumer contract, run the following command in the terminal. It will execute a script that deploys the compiled contract to the Kovan test network

```
brownie run scripts/vrf_scripts/01_deploy_vrf.py --network kovan
```



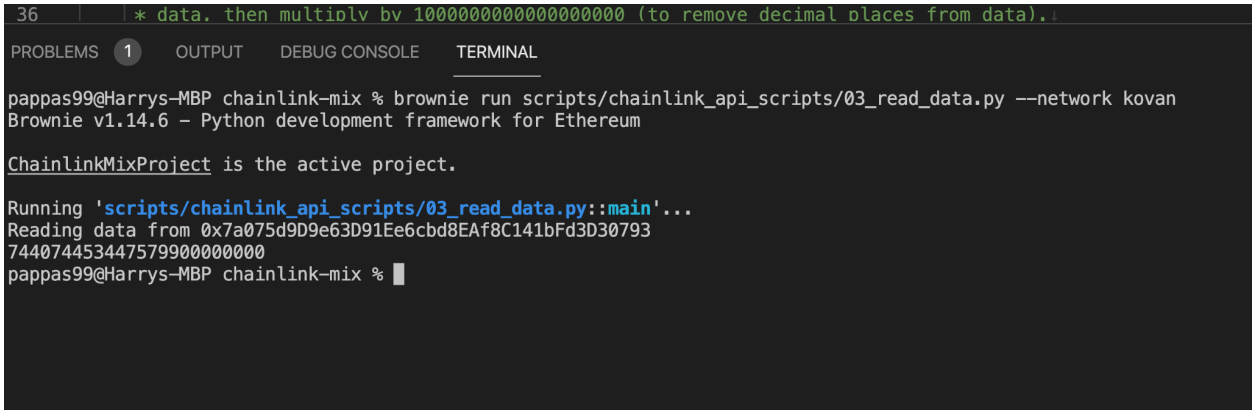
8. To interact with the deployed VRFConsumer contract, run the '02_request_randomness.py' script. This script will fund the deployed contract with some LINK from your MetaMask wallet, and then it will execute the getRandomNumber function in the contract, which will reach out to a Chainlink Oracle and make a request for randomness

```
brownie run scripts/vrf_scripts/02_request_randomness.py --network kovan
```



9. To see the result of the randomness request in your smart contract, run the '03_read_random_number.py' script. Ensure a few seconds has passed since you ran the previous '02_request_randomness' script, to ensure the Chainlink Oracle has successfully made the callback transaction

```
brownie run scripts/vrf_scripts/03_read_random_number.py --network kovan
```



You have now successfully performed a request for randomness using Chainlink VRF

Congratulations, you've successfully used the Brownie Starter Kit to perform the following:

- Deploy three smart contracts to the Kovan testnet
- Interacted with a deployed smart contract to use Chainlink Data Feeds
- Interacted with a deployed smart contract to perform an API request using a Chainlink Oracle
- Interacted with a deployed smart contract to perform a request for randomness using Chainlink VRF

Bonus Exercises:

You can attempt to complete these exercises if you've completed the main exercise ahead of schedule:

1. Create a new contract in the contracts folder of the Brownie Starter Kit. You can create or put any contract inside it! If you want some examples, check out the Solidity docs examples.

2. Create a script in the scripts folder to deploy your contract. Use the existing deploy scripts as a guide for the syntax.

3. Compile and deploy your contracts to the kovan network by executing the script you created previously.

```
brownie compile
```

```
brownie run script-name-goes-here.py --network kovan
```

Create a second script to interact with your deployed contract, calling it's functions. Use the existing scripts that interact with contracts as a guide for the syntax that's needed. Execute your script in the terminal, and see the output.

```
brownie run script-name-goes-here.py --network kovan
```

Exercise 3: Deploying to a Local Network

In this exercise, you'll deploy the Brownie Starter Kit smart contracts to a local Ganache network, and interact with them.

Brownie's built in development network

Brownie has 2 types of local networks:

Those of type `Ethereum` and those of type `Development`.

`Ethereum`: These are networks that brownie will keep track of all deployments in the build folder.
`Development`: These networks will not keep track of deployments, and will be spun up and torn down for every script.

Viewing Your Networks

1. Open Visual Studio Code, ensure your current project is the 'chainlink-mix' workspace from day 2 of the Developer Bootcamp (hint: If it's not, go File -> Open, and select the 'chainlink-mix' folder)

2. If it's not already open, open the VS Terminal (View -> Terminal or CTRL + `)

3. If you're not already in it, head into your 'chainlink-mix' folder in the terminal
    a. Or, you can redownload it with:

```
brownie bake chainlink-mix
cd chainlink
```

4. Once you're in the mix folder, we can view all the different networks we can connect to by running:

```
brownie networks list
```

And you'll see an output like:

```
Brownie v1.14.6 - Python development framework for Ethereum

The following networks are declared:

Ethereum
  ├─Mainnet (Infura): mainnet
  ├─Ropsten (Infura): ropsten
  ├─Rinkeby (Infura): rinkeby
  ├─Goerli (Infura): goerli
  ├─Kovan (Infura): kovan
  ├─sokol: sokol
  ├─mumbai: mumbai
  ├─matic-fork: matic-fork
  ├─binance-smart-chain: binance-smart-chain
  ├─matic: matic
  ├─matic-mainnet: matic-mainnet
  ├─xDai: xDai
  ├─binance-smart-chain2: binance-smart-chain2
  ├─ganache: ganache
  ├─local: local
  ├─fuji: fuji
  ├─forked-mainnet: forked-mainnet
  ├─ganache-forked: ganache-forked
  └─local-ganache: local-ganache

Ethereum Classic
  ├─Mainnet: etc
  └─Kotti: kotti

Development
  ├─Ganache-CLI: development
  ├─binance-fork: binance-fork
  ├─hardhat: hardhat
  ├─rinkeby-fork: rinkeby-fork
  └─mainnet-fork: mainnet-fork
```

You may not have all the same networks here. You can ignore the “Ethereum Classic” networks for now.

This is a list of all the networks we can connect to. If you type:

```
brownie networks list True
```

You’ll get an output that shows all the parameters for connecting to those networks.

Running the default development network

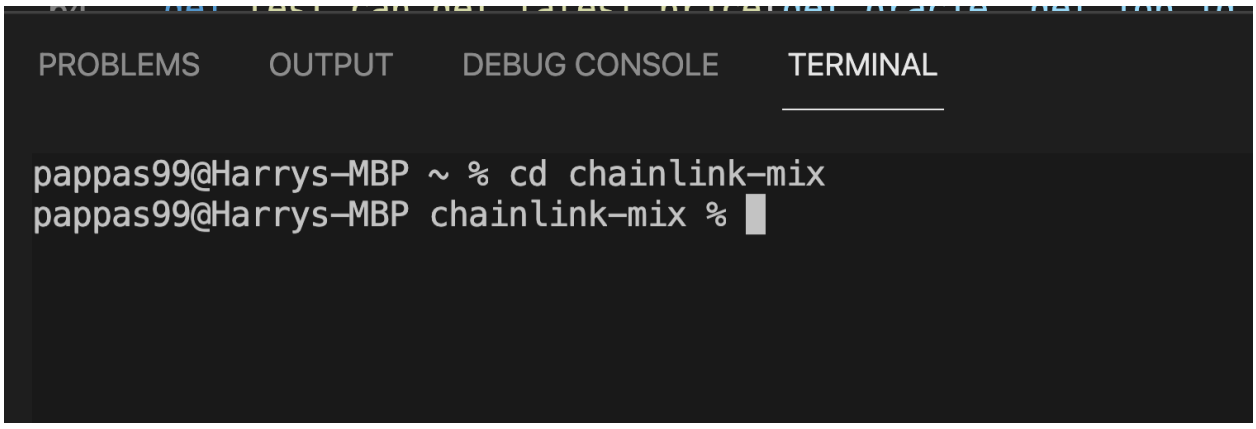
When we run a script, we have a default network that is connected if we don’t tell brownie which network to use. The default network is the `development` network. We can run a script and connect to this temporary network by running a script, and not choosing a network.

Setting up the Local Development Ganache Network

1. Open Visual Studio Code, ensure your current project is the ‘chainlink-mix’ workspace from day 2 of the Developer Bootcamp (hint: If it’s not, go File -> Open, and select the ‘chainlink-mix’ folder)

2. If it’s not already open, open the VS Terminal (View -> Terminal or CTRL + `)

3. If you’re not already in it, head into your ‘chainlink-mix’ folder in the terminal



5. The first thing you need to do is download and install Ganache. Run the following command. You can skip this step if you installed it during the last session for your project. **Remember, you’ll need nodejs installed from the** setup Instructions**.**

```
npm install -g ganache-cli
```

If you get an error about ‘missing write access’, try running it with sudo in front of it

```
sudo npm install -g ganache-cli
```

6. Next, run a ganache local blockchain by running the command. You can do this in a new terminal window in VS Code if you like.

```
ganache-cli
```

You’ll see an output like:

```
Ganache CLI v6.12.2 (ganache-core: 2.13.2)

Available Accounts
==================
(0) 0x3060571cA9fF5529CA3b506F8E5A4430E7529006 (100 ETH)
(1) 0xAE0d6b6b509ebf7945F1898D6284e5d98ab5eE94 (100 ETH)
(2) 0x8e28474145eE9E09970c0C1C6b22e679BDd1BC81 (100 ETH)
(3) 0x9a4CCFCf20fe4d6a39d4668B93ff6e461246bEf9 (100 ETH)
(4) 0x035d52C1399Bb6ee7E0EC0e322F32ac1806678a0 (100 ETH)
(5) 0xAF01f6e7Ac2972a892638B7c48A0D1f34cB166F5 (100 ETH)
(6) 0x394bb4F7983eb23952B7B1c5cffDd8E7028Fd5CB (100 ETH)
(7) 0xf312e601a5EdD3B79Fc414B416f3ED405D7fc034 (100 ETH)
(8) 0x367140b3D87Cd1EEDD71A0D970087a7d2671086E (100 ETH)
(9) 0x59AFDb5c40751A72Ee50Dd2B1759Ca6ba5763309 (100 ETH)

Private Keys
==================
(0) 0x31ccb3c5133784d834dc2036281ab04906621a8a258548969d595d931d35717a
(1) 0x74d40321a4dbdb5b0af2d70c0132fa51eeb85a92c2096e7084a4c7c05a3aaab5
(2) 0xd208ba7f63bcd4b95a60298bfb046ac12c80e332ee85de514409cdb526aff746
(3) 0xe9d0684e5f041aa3e228f9cb764f4a5d6a8920682ed490376a1d926e144950db
(4) 0xe57eb35c6ec64222ffec673815c614c8dbc9d59c458484dea3cad71e41e078e3
(5) 0xa67ed4a39a2693f844aa067a18bbbbe25bec867aa47db91ef042bdada14d6e3d
(6) 0x9c3029c56963be6b684797a81709cdcc5f21fd7e63dd83e413a9b2c2ee0c2a71
(7) 0x24b33f1502f237583c6a5e2ca4eb193ff581e04039c9ecdeec8b90b609f514e5
(8) 0x2297dc7a0db1a1586c2676d2270e5698839e745ba183cbfdc4b2e5ddf33421a1
(9) 0x6010cd4aef2fb2306e26554a22b96ced97c7cc4c6ed158b6b3472e75ab604577
```

```
HD Wallet
==================
Mnemonic:      large reopen story nose exclude coach truck order purse
glide mandate round
Base HD Path:  m/44'/60'/0'/0/{account_index}

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

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

Call Gas Limit
==================
9007199254740991

Listening on 127.0.0.1:8545
```

7. Next you need to add a new network to the Brownie config so that it knows about your local Ganache instance. Run the following command. If you get an error about Ganache already existing, you can ignore it and move on.

```
brownie networks add Ethereum ganache host=http://localhost:8545
chainid=1337
```

Now we have 2 choices of where we want to deploy. We can deploy our scripts from the **persistent ganache network,** or our **development temporary ganache network.**

The development network is network **development,** and the persistent one is **ganache.** Let's change our default network to ganache. **This means that we will need our ganache-cli chain to always be running.**

Let’s change our brownie-config.yaml to look like:

```
networks:
  default: ganache
```

Deploying and Interacting With the Smart Contracts

Price Feeds Consumer Contract

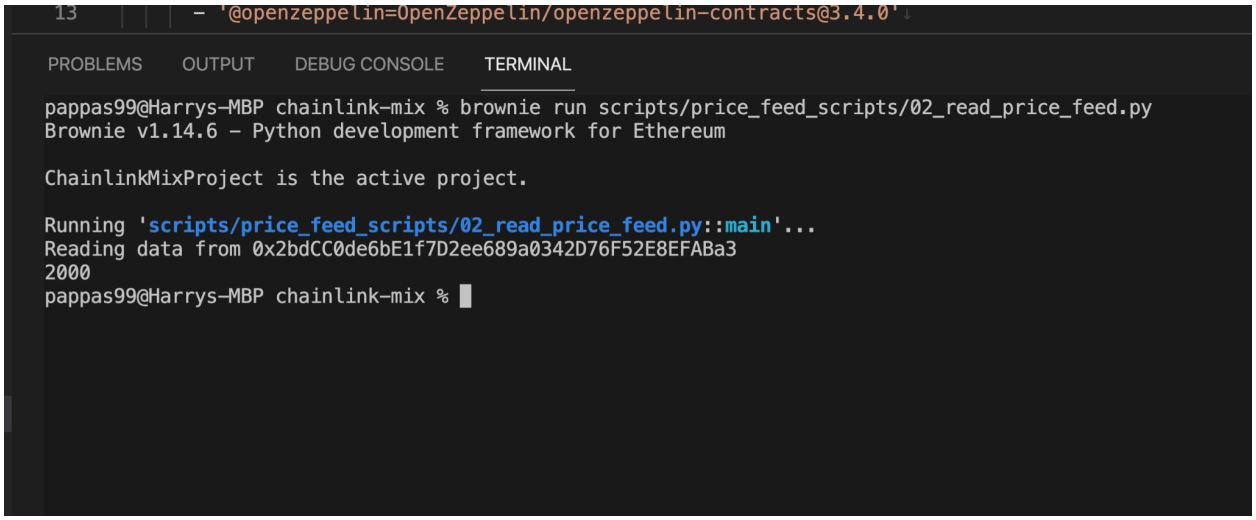
To deploy the PriceFeedConsumer contract to your local Ganache network, run the following command in the terminal. It will execute a script that deploys the compiled contract to your local Ganache network, along with some other mock scripts that mock up interaction with Chainlink Oracles and smart contracts.

```
brownie run scripts/price_feed_scripts/01_deploy_price_consumer_v3.py
```

```
pappas99@Harrys-MBP chainlink-mix % brownie run scripts/price_feed_scripts/01_deploy_price_consumer_v3.py
Brownie v1.14.6 - Python development framework for Ethereum

ChainlinkMixProject is the active project.

Running 'scripts/price_feed_scripts/01_deploy_price_consumer_v3.py::main'...
The active network is ganache
Deploying Mocks...
Deploying Mock Link Token...
Transaction sent: 0x66fca472240a5244a94a5c46370251eab7aef2ec187af77c0bc04cfe2fbd896a
  Gas price: 8.0 gwei   Gas limit: 736049   Nonce: 51
  LinkToken.constructor confirmed - Block: 12719755   Gas used: 669136 (90.91%)
  LinkToken deployed at: 0x5f3f1dBD7B74C6B46e8c44f98792A1dAf8d69154

Deploying Mock Price Feed...
Transaction sent: 0x7eb867f0970b0620c67f29522d96e9b23cc6f6f30bcfeeffb2e71a3f850c612c
  Gas price: 8.0 gwei   Gas limit: 473698   Nonce: 52
  MockV3Aggregator.constructor confirmed - Block: 12719756   Gas used: 430635 (90.91%)
  MockV3Aggregator deployed at: 0xb7278A61aa25c888815aFC32Ad3cC52fF24fE575

Deployed to 0xb7278A61aa25c888815aFC32Ad3cC52fF24fE575
Deploying Mock VRFCoordinator...
Transaction sent: 0x94036c39a75697743b0e8c42e59d2267237830a42ea0398da48d9b2a6100bcc4
  Gas price: 8.0 gwei   Gas limit: 304034   Nonce: 53
  VRFCoordinatorMock.constructor confirmed - Block: 12719757   Gas used: 276395 (90.91%)
  VRFCoordinatorMock deployed at: 0xCD8a1C3ba11CF5ECfa6267617243239504a98d90

Deployed to 0xCD8a1C3ba11CF5ECfa6267617243239504a98d90
Deploying Mock Oracle...
Transaction sent: 0xeff8c95f5391510155ae423b66ea6f3ba11e379408befcfecb562ebdf1fa1350
  Gas price: 8.0 gwei   Gas limit: 802385   Nonce: 54
  MockOracle.constructor confirmed - Block: 12719758   Gas used: 729441 (90.91%)
  MockOracle deployed at: 0x82e01223d51Eb87e16A03E24687EDF0F294da6f1

Deployed to 0x82e01223d51Eb87e16A03E24687EDF0F294da6f1
Mocks Deployed!
Transaction sent: 0x18adc9af3ce927a12d6f2c438f0c0166132c6cc6fa86bb4fbdabd5df922fc78c
  Gas price: 8.0 gwei   Gas limit: 145600   Nonce: 55
  PriceFeedConsumer.constructor confirmed - Block: 12719759   Gas used: 132364 (90.91%)
  PriceFeedConsumer deployed at: 0x2bdCC0de6bE1f7D2ee689a0342D76F52E8EFABa3

The current price of ETH is 2000
pappas99@Harrys-MBP chainlink-mix %
```

To interact with the deployed Price Feed Consumer contract, run the '02_read_price_feed.py' script, which will query the deployed smart contract, and return the latest price of the specified price feed. You will notice the value is set to a static amount. This is because our local network is a fresh blockchain without any data, so we're just mocking up a Price Feed Aggreagator contract.

```
brownie run scripts/price_feed_scripts/02_read_price_feed.py
```



You have now successfully used Chainlink Data Feeds in your smart contract running on a local Ganache network.

Forking Ethereum Mainnet

1. Now that you've got a local Ganache network working, let's modify it and change its initial state to be a fork of the current Ethereum mainnet, and then deploy and interact with a Price Feed Consumer contract. First, you should tell your brownie config to use it as the default network. Update the brownie-config.yaml file, and set the default network to mainnet-fork:

   ```
   networks:
     default: mainnet-fork
   ```

   If you scroll down in the config file, you will see that mainnet-fork network uses a specific eth_usd_price_feed address. If you head over to the Chainlink documentation, you will see that this is the contract address for the Ethereum mainnet ETH/USD Price Feed

```
    fee: 100000000000000000
  binance:
    # link_token: ??
    eth_usd_price_feed: '0x9ef1B8c0E4F7dc8bF5719Ea496883DC6401d5b2e'
  binance-fork:
    eth_usd_price_feed: '0x9ef1B8c0E4F7dc8bF5719Ea496883DC6401d5b2e'
  mainnet-fork:
    eth_usd_price_feed: '0x5f4eC3Df9cbd43714FE2740f5E3616155c5b8419'
  matic-fork:
    eth_usd_price_feed: '0xF9680D99D6C9589e2a93a78A04A279e509205945'
wallets:
  from_key: ${PRIVATE_KEY}
  from_mnemonic: ${MNEMONIC}
```

You're now ready to deploy your price feed consumer contract to a temporary local Ganache network running as a fork of Ethereum mainnet!

2. Stop your `ganache-cli` chain. You can end it by hitting "Ctrl + C"

```
====================
20000000000

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

Call Gas Limit
==================
9007199254740991

Listening on 127.0.0.1:8545
^Cpatrick@iMac: [~] $
```

3. To deploy the PriceFeedConsumer contract to your local forked Ganache network, run the following command in the terminal. It will execute a script that deploys the compiled contract to your local Ganache network. Take note of the listed block number, it indicates the exact point in time at which the Ethereum mainnet network was forked. You can look up the block number at https://etherscan.io/.

```
brownie run scripts/price_feed_scripts/01_deploy_price_consumer_v3.py
```

```
pappas99@Harrys-MBP chainlink-mix % brownie run scripts/price_feed_scripts/01_deploy_price_consumer_v3.py
Brownie v1.14.6 - Python development framework for Ethereum

ChainlinkMixProject is the active project.
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/brownie/network/main.py:44: BrownieEnvironmentWarning: Development network has a block height of 12719765
  warnings.warn(
Attached to local RPC client listening at '127.0.0.1:8545'...

Running 'scripts/price_feed_scripts/01_deploy_price_consumer_v3.py::main'...
Transaction sent: 0xbe36b2bd58bb82e526151e573e8e809c9cb2a4394a40ded10d6ab600ef84c188
  Gas price: 0.0 gwei   Gas limit: 9500000   Nonce: 62
  PriceFeedConsumer.constructor confirmed - Block: 12719766   Gas used: 132364 (1.39%)
  PriceFeedConsumer deployed at: 0xB0D4afd8879eD9F52b28595d31B441D079B2Ca07

The current price of ETH is 197358000000
pappas99@Harrys-MBP chainlink-mix %
```

In the output above, you can see the currently returned price of ETH is an actual value this time, and not a static amount. If you go to the page for the ETH/USD feed on Ethereum mainnet, you should see a similar result.

Note: You will not be able to run 02_read_price_feed.py… Can you guess why?

Congratulations, you've successfully used the Brownie Starter Kit to perform the following:

1. Deploy smart contracts to a local network
2. Interacted with deployed smart contracts and mock contracts on a local network
3. Created a fork of the Ethereum mainnet on a local network, deployed smart contracts to it, and interacted with the mainnet ETH/USD price feed.

Bonus Exercises:

You can attempt to complete these exercises if you've completed the main exercise ahead of schedule:

1. If you haven't completed the bonus exercises from earlier, create a new contract in your contracts folder called 'ERC-20.sol', paste the following code in it and save your work. This is a simple ERC-20 token contract. You can modify the symbol and name parameters if you wish. If you have already this, you can skip this step

```
pragma solidity ^0.6.0;

interface IERC20 {

    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns
(uint256);
```

```
    function allowance(address owner, address spender) external
view returns (uint256);

    function transfer(address recipient, uint256 amount)
external returns (bool);
    function approve(address spender, uint256 amount) external
returns (bool);
    function transferFrom(address sender, address recipient,
uint256 amount) external returns (bool);


    event Transfer(address indexed from, address indexed to,
uint256 value);
    event Approval(address indexed owner, address indexed
spender, uint256 value);
}


contract ERC20Basic is IERC20 {

    string public constant name = "ERC20Basic";
    string public constant symbol = "ERC";
    uint8 public constant decimals = 18;


    //event Approval(address indexed tokenOwner, address
indexed spender, uint tokens);
    //event Transfer(address indexed from, address indexed to,
uint tokens);


    mapping(address => uint256) balances;

    mapping(address => mapping (address => uint256)) allowed;

    uint256 totalSupply_;

    using SafeMath for uint256;


   constructor(uint256 total) public {
    totalSupply_ = total;
    balances[msg.sender] = totalSupply_;
    }
```

```
    function totalSupply() public override view returns
(uint256) {
    return totalSupply_;
    }

    function balanceOf(address tokenOwner) public override view
returns (uint256) {
        return balances[tokenOwner];
    }

    function transfer(address receiver, uint256 numTokens)
public override returns (bool) {
        require(numTokens <= balances[msg.sender]);
        balances[msg.sender] =
balances[msg.sender].sub(numTokens);
        balances[receiver] = balances[receiver].add(numTokens);
        emit Transfer(msg.sender, receiver, numTokens);
        return true;
    }

    function approve(address delegate, uint256 numTokens)
public override returns (bool) {
        allowed[msg.sender][delegate] = numTokens;
        emit Approval(msg.sender, delegate, numTokens);
        return true;
    }

    function allowance(address owner, address delegate) public
override view returns (uint) {
        return allowed[owner][delegate];
    }

    function transferFrom(address owner, address buyer, uint256
numTokens) public override returns (bool) {
        require(numTokens <= balances[owner]);
        require(numTokens <= allowed[owner][msg.sender]);

        balances[owner] = balances[owner].sub(numTokens);
        allowed[owner][msg.sender] =
allowed[owner][msg.sender].sub(numTokens);
        balances[buyer] = balances[buyer].add(numTokens);
        emit Transfer(owner, buyer, numTokens);
        return true;
    }
}
```

```
library SafeMath {
    function sub(uint256 a, uint256 b) internal pure returns
(uint256) {
      assert(b <= a);
      return a - b;
    }

    function add(uint256 a, uint256 b) internal pure returns
(uint256) {
      uint256 c = a + b;
      assert(c >= a);
      return c;
    }
}
```

2. Create a deploy script for the new contract in your deploy folder. Call it deploy_erc20.py. If you already created this in the last session's bonus exercises, you can skip this step. This script will deploy your ERC-20 token.

```
from brownie import ERC20Basic, config, accounts

def deployContract():
    account = accounts.add(config["wallets"]["from_key"])
    ERC20Basic.deploy(100000000,{'from': account})

def main():
    deployContract()
```

3. Create a script to interact with your deployed ERC20 contract. Call it interact_erc20.py. This script will call a function to check to see what the total supply of the token is.

```
from brownie import ERC20Basic, config, accounts, network

def main():
    account = accounts.add(config["wallets"]["from_key"])
    erc20 = ERC20Basic[-1]
    print("Total Supply is", erc20.totalSupply())
```

4. Deploy your new contract to your local network, then run the interact script to see the result

```
brownie run scripts/deploy_erc20.py
```

```
brownie run scripts/interact_erc20.py
```

5. Modify your interact script to call other functions in the ERC-20 contract, such as to Transfer funds (transfer function) from your current wallet address to another one that you own on the local Ganache instance. Remember you can run 'ganache-cli' at the terminal to see all of the wallet addresses you control, and send tokens from your main (first) account, to another one such as your second account. You can stop your Ganache instance from running by typing CTRL + C in the terminal that's running Ganache.

```
pappas99@Harrys-MBP tests % ganache-cli
Ganache CLI v6.12.2 (ganache-core: 2.13.2)

Available Accounts
==================
(0) 0x3Cd84fCF792a05095A5db47ed64CbD9a16b24846 (100 ETH)
(1) 0xBDB62d426FB2D7AaA7C9a1bB09CF969C09AA42e7 (100 ETH)
(2) 0x46C256Da71C67ce72b2ff26b527C4E156fbF7582 (100 ETH)
(3) 0x9e922714a098D6Af53fEBF1c6547272F06eE1057 (100 ETH)
(4) 0x1AdD9319b8d3E552F4244F14E53aC1E15f8347D5 (100 ETH)
(5) 0x73CE0a1417a22aB0a4ee91EDe2e01385Aae33a13 (100 ETH)
(6) 0x2C1A19c1d055257eAc894a2E85fbAC70E1e9cc2c (100 ETH)
(7) 0x9d007E74081F0922c391Cf22b3B2A6632B669e59 (100 ETH)
(8) 0x4EB6504Cf6d440B024A7Dfa15E266B5049c95108 (100 ETH)
(9) 0xCCd829845D90a77Fa2AA9B89D9D58974F815315C (100 ETH)
```

Exercise 4: Testing Smart Contracts

In this exercise, you'll execute the tests that come with the Brownie Starter Kit, do a Solidity Coverage test and then check in the project to a public repository on GitHub!

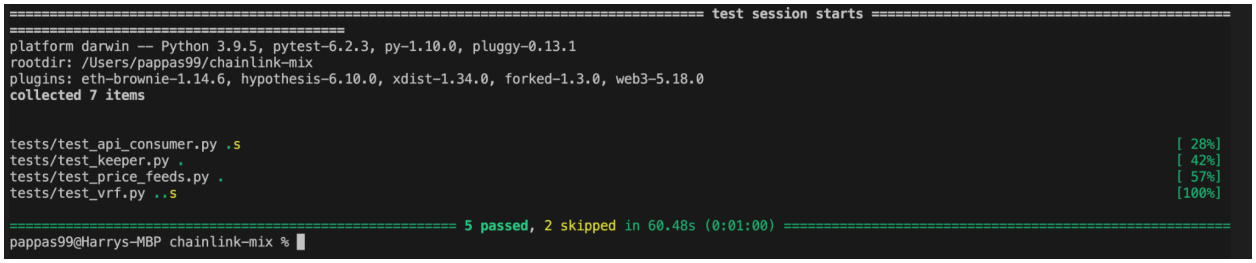
Executing the Unit Tests

1. First thing you need to do is to change the default network back to the local development network in the brownie-config.yaml file

```
networks:
  default: development
```

2. Now you're going to execute the unit tests. These will run a few basic tests against the smart contracts, and use mocks to simulate connecting to an external oracle network. Run the following command in the console to execute the unit tests

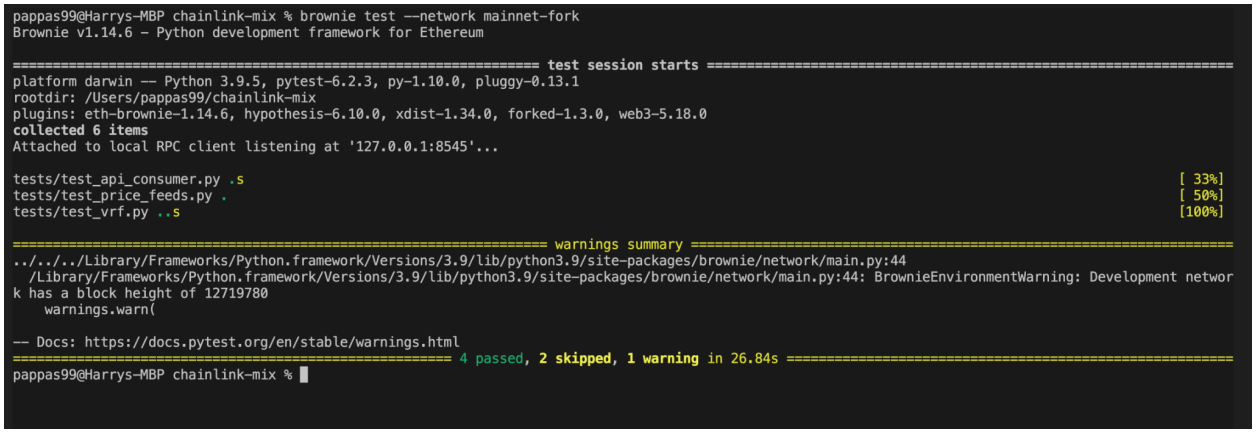
```
brownie test
```



You should see all unit tests pass successfully.

3. Now lets execute the tests again, but this time on a fork of the Ethereum mainnet. To run them again on a forked network, execute the following command

```
brownie test --network mainnet-fork
```

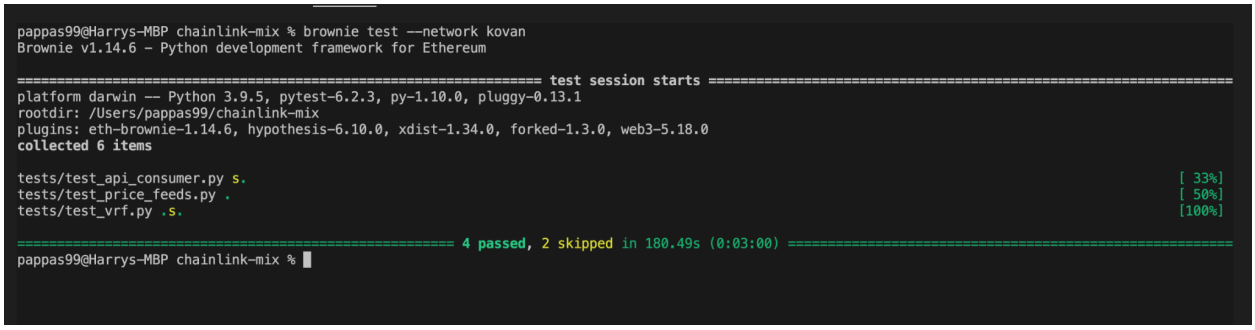


Next you'll run the tests on the public Kovan test network to test the actual integration of our smart contracts to the Chainlink Oracle network.

Executing the Integration Tests

You can now execute the integration tests on the Kovan network by running the following command. The integration tests may take a couple minutes to complete.

```
brownie test --network kovan
```



Checking the Solidity Coverage

The last step in your testing is to check what the Solidity coverage is in your tests.

1. To check your unit tests coverage, execute the following command

```
brownie test --coverage
```

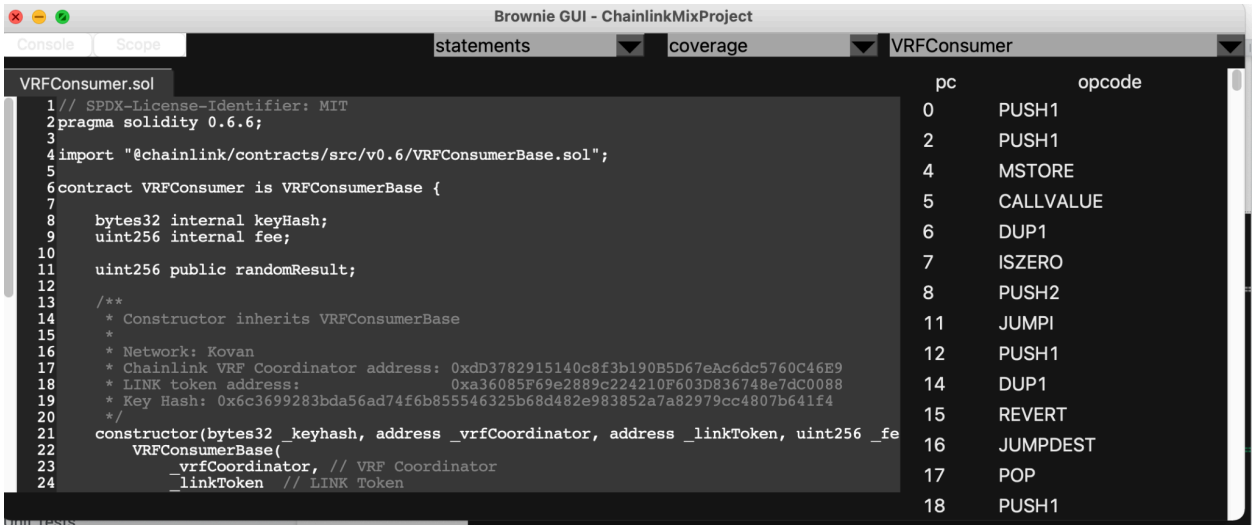
```
pappas99@Harrys-MBP chainlink-mix % brownie test --coverage
Brownie v1.14.6 - Python development framework for Ethereum

============================= test session starts ==============================
platform darwin -- Python 3.9.5, pytest-6.2.3, py-1.10.0, pluggy-0.13.1
rootdir: /Users/pappas99/chainlink-mix-3/chainlink-mix
plugins: eth-brownie-1.14.6, hypothesis-6.10.0, xdist-1.34.0, forked-1.3.0, web3-5.18.0
collected 7 items
Attached to local RPC client listening at '127.0.0.1:8545'...

tests/test_api_consumer.py .s                                            [ 28%]
tests/test_keeper.py .                                                   [ 42%]
tests/test_price_feeds.py .                                              [ 57%]
tests/test_vrf.py ..s                                                    [100%]

============================= warnings summary ==============================
../../../../Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/brownie/network/main.py:44
  /Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/brownie/network/main.py:44: BrownieEnvironmentWarning: Development network has a block height of 12747227
    warnings.warn(

-- Docs: https://docs.pytest.org/en/stable/warnings.html
================================ Coverage ================================

  contract: APIConsumer - 59.5%
    APIConsumer.fulfill - 100.0%
    APIConsumer.requestVolumeData - 100.0%
    BufferChainlink.append - 100.0%
    BufferChainlink.appendInt - 100.0%
    BufferChainlink.appendUint8 - 100.0%
    CBORChainlink.encodeString - 100.0%
    Chainlink.add - 100.0%
    Chainlink.addInt - 100.0%
    Chainlink.initialize - 100.0%
    ChainlinkClient.buildChainlinkRequest - 100.0%
    ChainlinkClient.encodeRequest - 100.0%
    ChainlinkClient.sendChainlinkRequestTo - 75.0%
    BufferChainlink.write - 66.7%
    CBORChainlink.encodeType - 62.8%
    BufferChainlink.init - 50.0%
    BufferChainlink.writeInt - 50.0%
    BufferChainlink.writeUint8 - 50.0%
    CBORChainlink.encodeInt - 37.5%
```

Ln 20, Col 23 Spaces: 2 UTF-8 LF YAML

A report should be displayed, showing how much of the code is being tested in your tests (see the percentages in yellow). You can view the test coverage data in the brownie gui

```
brownie gui
```

You can see the report for each contract there

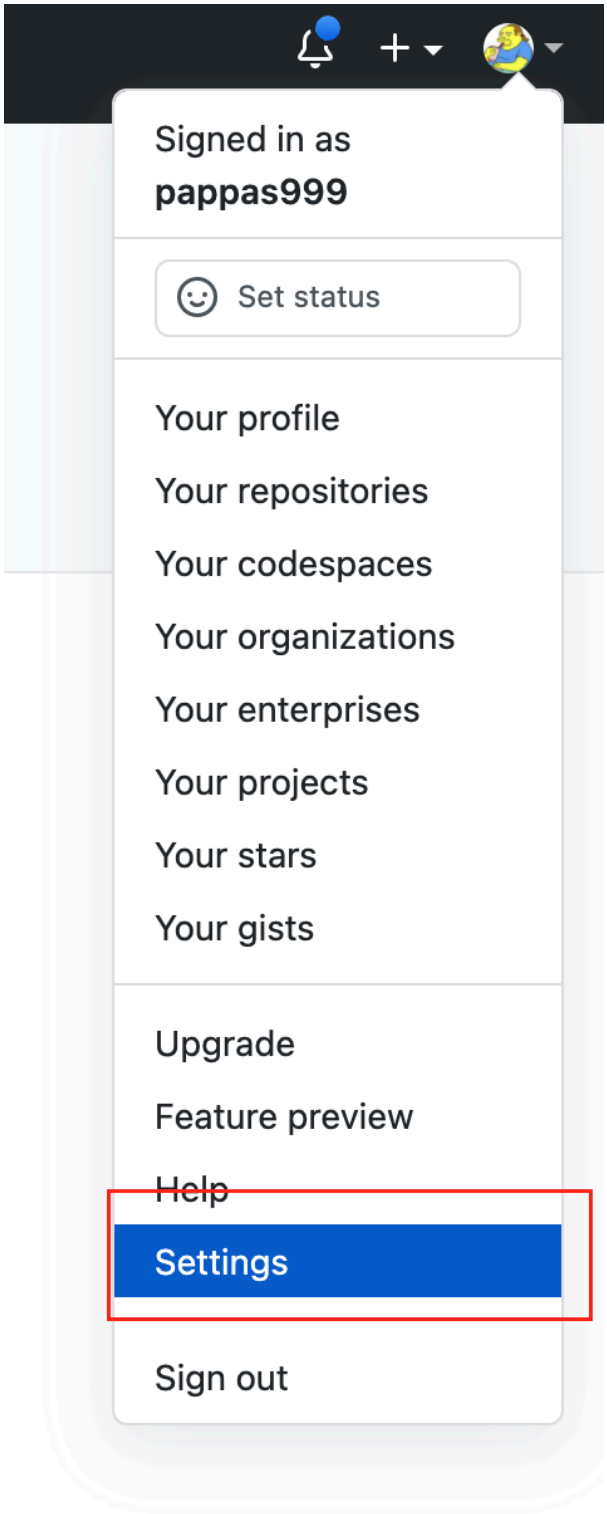


Congratulations, you've learned how to execute unit tests on a local ganache network as well as a forked local network, how to execute integration tests on a public network, and how to check the coverage of your unit tests. Now the last step is checking your project into a public GitHub repository so you can share it with the world!

Checking in your Project

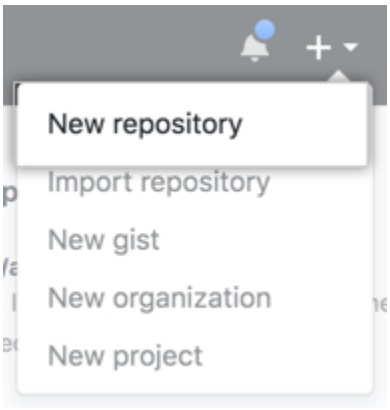
Perform the following steps to check in your brownie project into a public repository for everyone else to see your work

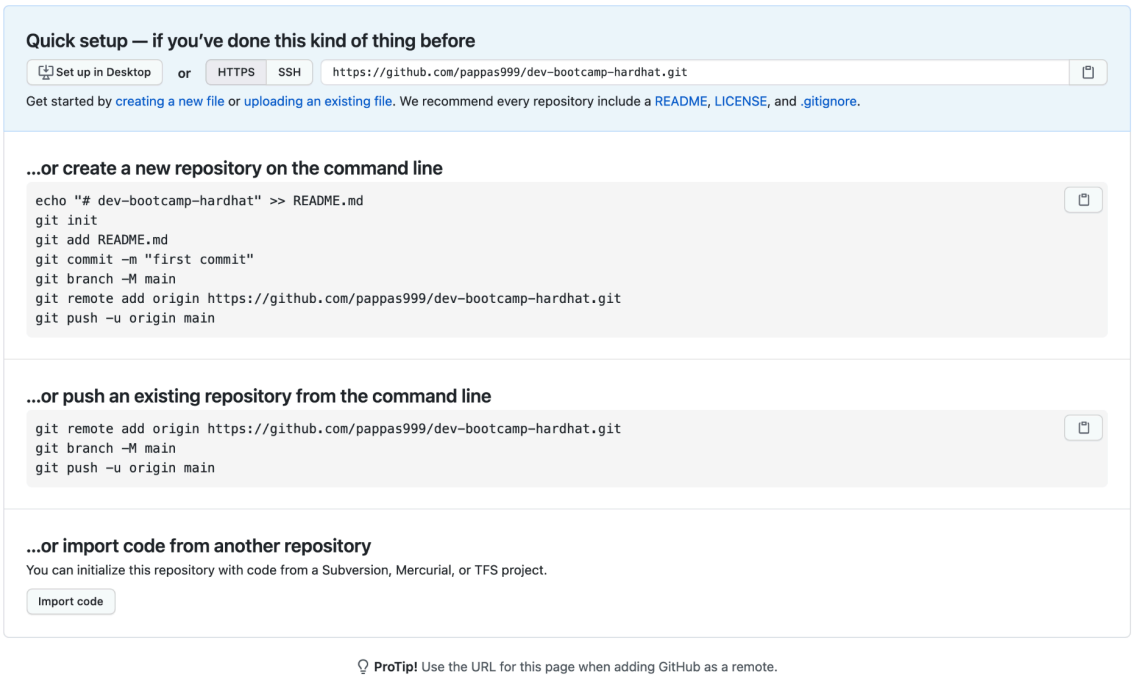
1. Head over to https://github.com/ and sign-in, or create a new free account if you don't have one already.
   Click on your profile on the top-right corner, and goto Settings -> Developer Settings -> Personal access tokens



2. Generate a personal access token as per the GitHub instructions

3. Create a new repository on GitHub. Call it ‘dev-bootcamp-brownie’, and leave the visibility as ‘public’. To avoid errors, do not initialize the new repository with README, license, or gitignore files. Once you get to the ‘setup page’, leave it open, you will come back to it soon.





4. Go back to your VS Code terminal for your project.

5. Initialize the local directory as a Git repository.

```
git init -b master
```

6. Add the files in your new local repository. This stages them for the first commit.

```
git add .
```

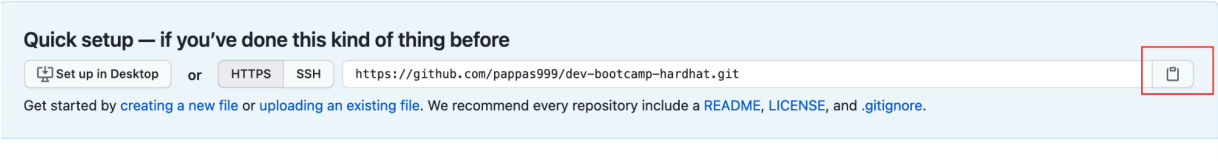
7. Next we want to remove the existing remote link to the Brownie Starter Kit repository on GitHub

```
git remote rm origin
```

8. Commit the files that you've staged in your local repository.

```
git commit -m "Developer Bootcamp".
```

9. Go back to GitHub. At the top of your GitHub repository's Quick Setup page, click the copy button to copy the remote repository URL.



10. Back in VS code terminal, you need to add the URL for the remote repository where your local repository will be pushed. Enter in the command below, and replace the <PASTE_REMOTE_URL> with the value you copied above. If prompted for GitHub authentication details, use your email that you signed up to GitHub with, and your personal access token that you generated earlier as the password. If you don't get prompted, you can continue.

```
git remote add origin  <PASTE_REMOTE_URL>
```

11. Verify the new remote URL with the following command. You should see it now pointing to your GitHub repository. This means when we push code, we will be pushing it to your GitHub repository.

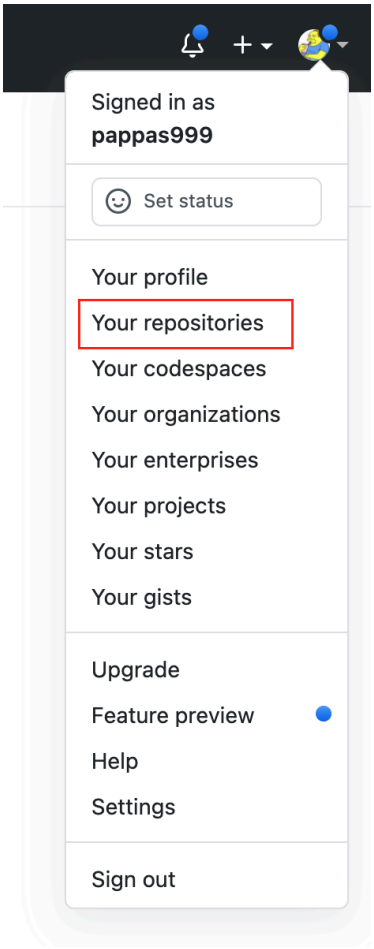
```
git remote -v
```

```
pappas99@Harrys-MBP contracts % git remote -v
origin  https://github.com/pappas999/dev-bootcamp-brownie.git (fetch)
origin  https://github.com/pappas999/dev-bootcamp-brownie.git (push)
pappas99@Harrys-MBP contracts %
```

12. Push the changes in your local repository up to GitHub. If prompted for GitHub authentication details, use your email that you signed up to GitHub with, and your personal access token that you generated earlier as the password. This will give your local git program access to push code up to your GitHub account.

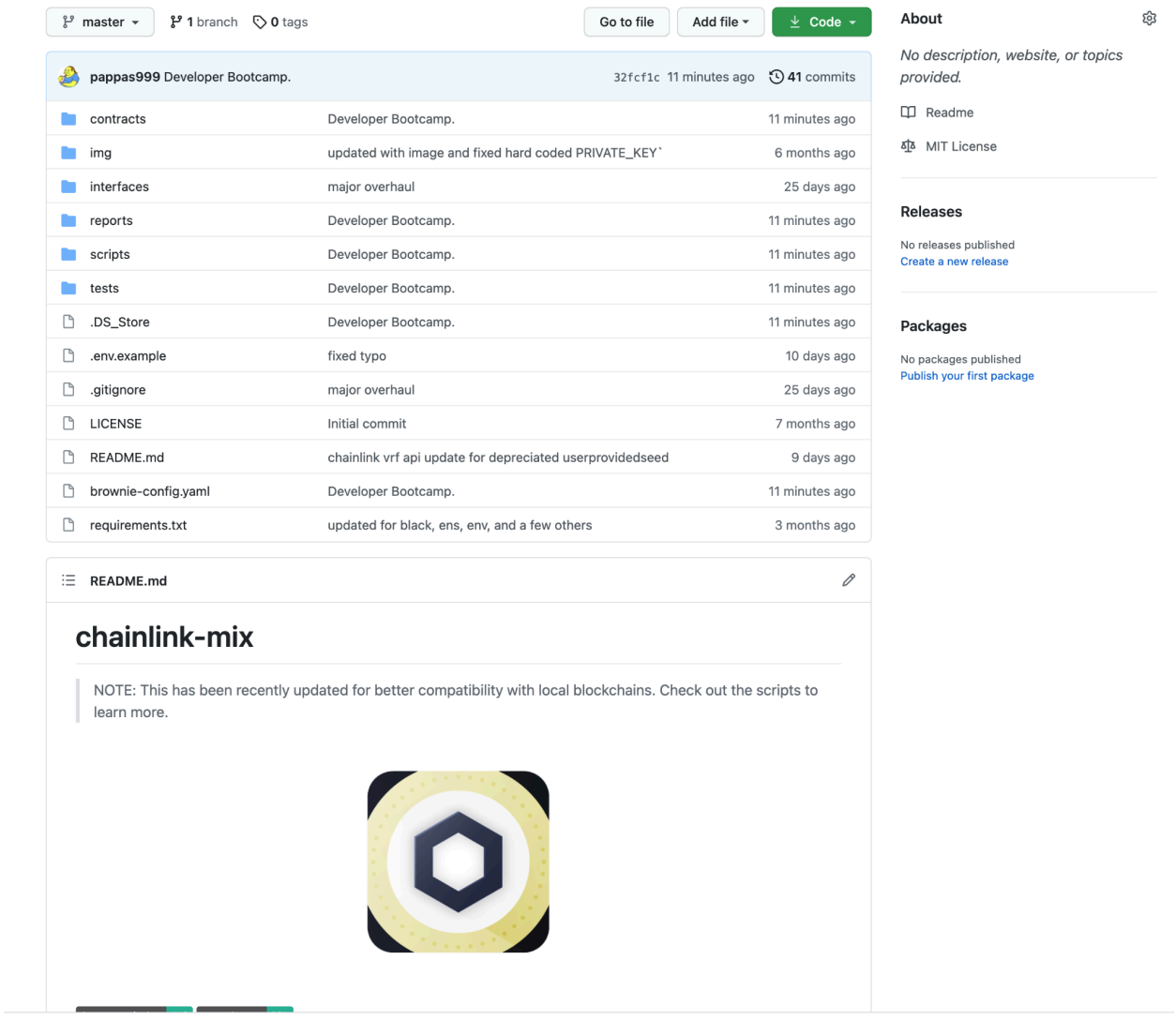
```
git push -u origin master
```

Your repository should now be live on GitHub! If you still have GitHub open with your new repository, then you can just refresh the page. Otherwise, head to GitHub, then in the top right corner open the menu and choose 'your repositories', then click on the URL link to your project to open it up and see. Take note of the URL and share it in the Bootcamp chat to share your completed project with everyone, or share it on Twitter with the hashtag **#chainlink-dev-bootcamp**, or in the **#developer-bootcamp** channel in the Chainlink Discord!



If you've made it this far, congratulations, you've completed the Summer 2021 Smart Contract Developer Bootcamp!

If you finished early, you can edit the README file and modify the text to something more appropriate. You can do this directly in GitHub by pressing on the edit icon, then modifying the file and saving your changes. You can also edit the project about/description directly in GitHub



Bonus Exercise:

You can attempt to complete these exercises if you've completed the main exercise ahead of schedule. You can refer to previous exercise steps for a guide, or to use code as a template.

1. Create a new brownie project in a new folder, call it anything you like
2. Create a new smart contract inside a contracts folder. You can paste in a completed example, or come up with one on your own. You can even create one in Remix, then paste it into a new contract file in your brownie project once you are happy with it
3. Create a deployment script for your new contract, deploy your contract to Kovan and a local ganache instance
4. Create a unit test for your new contract, and execute it (hint: syntax is brownie test)
5. Check your new project into a new repository on GitHub

Appendix: Troubleshooting

Installing Pipx and Brownie

If you're using macOS and you get an error similar to this: *'invalid active developer path (/Library/Developer/CommandLineTools), missing xcrun at: /Library/Developer/CommandLineTools/usr/bin/xcrun'*, run the following command to download and install the apple developer command line tools, then re-open your terminal window and try the command again

```
xcode-select --install
```

If you still get errors, you can try install using pip instead of pipx

```
pip install eth-brownie
```

Windows Users:

If while installing pipx you get an error stating that 'pipx is not recognized as an internal or external command', ensure you're running VS Code or the terminal as administrator.

If you are following the setup instructions, try restarting windows after the 'python3 -m pipx ensurepath' command, before trying to install brownie

If you are able to successfully install brownie, but then get an error similar to the following: '*brownie' is not recognized as an internal or external command...*", try the following:

1. Install pipx

```
python3 -m pipx install --user pipx
```

2. Add "C:\Users\<replace-with-logged-on-userid>\ .local\bin" to your PATH windows system environment variable.

3. Try installing eth-brownie again, using pipx

```
pipx install eth-brownie
```

If that doesn't work, try adding your Python scripts folder to your PATH environment variable in windows. Eg

```
C:\Users\owner\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9
_qbz5n2kfra8p0\LocalCache\local-packages\Python39\Scripts
```

Then try reinstall brownie

Running Scripts

If you get an error specific to an environment variable such as WEB3_INFURA_PROJECT_ID or PRIVATE_KEY, try manually exporting them in your terminal, then try again. Ie copy the lines in your .env, and paste them in your terminal to run them as commands. Then try your command again

```
export WEB3_INFURA_PROJECT_ID='abcdef'
export PRIVATE_KEY='123456'
```

If you still can't get it working, if you're using windows, try adding the variable to your windows environment variables (in system settings), then try again