**Interacting with your Contracts**

Brownie has three main components that you can use while developing your project:

1. The [console](https://eth-brownie.readthedocs.io/en/v1.19.3/interaction.html#console) is useful for quick testing and debugging.
2. [Scripts](https://eth-brownie.readthedocs.io/en/v1.19.3/interaction.html#scripts) allow you to automate common tasks and handle deployments.
3. [Tests](https://eth-brownie.readthedocs.io/en/v1.19.3/interaction.html#tests-intro) help to ensure that your contracts are executing as intended.

**Using the Console**

The console is useful when you want to interact directly with contracts deployed on a non-local chain, or for quick testing as you develop. It’s also a great starting point to familiarize yourself with Brownie’s functionality.

The console feels very similar to a regular python interpreter. From inside a project directory, load it by typing:

$ brownie console

Brownie will compile the contracts, launch or attach to the local test environment, and then give you a command prompt. From here you may interact with the network with the full range of functionality offered by the [Brownie API](https://eth-brownie.readthedocs.io/en/v1.19.3/api.html#api).

**Hint**

You can call the builtin [**dir**](https://docs.python.org/3.8/library/functions.html#dir) method to see available methods and attributes for any class. Classes, methods and attributes are highlighted in different colors.

You can also call [**help**](https://docs.python.org/3.8/library/functions.html#help) on any class or method to view information on it’s functionality.

**Writing Scripts**

Along with the console, you can write scripts for quick testing or to automate common processes. Scripting is also useful when deploying your contracts to a non-local network.

Scripts are stored in the scripts/ directory within your project.

**Layout of a Script**

Brownie scripts use standard Python syntax, but there are a few things to keep in mind in order for them to execute properly.

**Import Statements**

Unlike the console where all of Brownie’s objects are already available, in a script you must first import them. The simplest way to do this is via a wildcard import:

**from** **brownie** **import** \*

This imports the instantiated project classes into the local namespace and gives access to the [Brownie API](https://eth-brownie.readthedocs.io/en/v1.19.3/api.html#api) in exactly the same way as if you were using the console.

Alternatively you may wish to only import exactly the classes and methods required by the script. For example:

**from** **brownie** **import** Token, accounts

This makes available the [**accounts**](https://eth-brownie.readthedocs.io/en/v1.19.3/api-network.html#brownie.network.account.Accounts) and [**Token**](https://eth-brownie.readthedocs.io/en/v1.19.3/api-network.html#brownie.network.contract.ContractContainer) containers, which is enough to deploy a contract.

**Functions**

Each script can contain as many functions as you’d like. When executing a script, brownie attempts to run the main function if no other function name is given.

**Running Scripts**

To execute a script from the command line:

$ brownie run <script> [function]

From the console, you can use the run method:

**>>>** run('token') *# executes the main() function within scripts/token.py*

You can also import and call the script directly:

**>>> from** **scripts.token** **import** main

**>>>** main()

**Examples**

Here is a simple example script from the token project, used to deploy the Token contract from contracts/Token.sol using web3.eth.accounts[0].

|  |  |
| --- | --- |
| 1  2  3  4 | **from** **brownie** **import** Token, accounts  **def** main():  Token.deploy("Test Token", "TST", 18, 1e23, {'from': accounts[0]}) |

And here is an expanded version of the same script, that includes a simple method for distributing tokens.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **from** **brownie** **import** Token, accounts  **def** main():  token = Token.deploy("Test Token", "TST", 18, 1e23, {'from': accounts[0]})  **return** token  **def** distribute\_tokens(sender=accounts[0], receiver\_list=accounts[1:]):  token = main()  **for** receiver **in** receiver\_list:  token.transfer(receiver, 1e18, {'from': sender}) |

**Writing Tests**

Brownie leverages pytest and hypothesis to provide a robust framework for testing your contracts.

Test scripts are stored in the tests/ directory of your project. To run the complete test suite:

$ brownie test

To learn more about writing tests in Brownie, you should start by reviewing the [Brownie Pytest documentation](https://eth-brownie.readthedocs.io/en/v1.19.3/tests-pytest-intro.html#pytest).

**Brownie Package Manager**

Brownie allows you to install other projects as packages. Some benefits of packages include:

* Easily importing and building upon code ideas written by others
* Reducing duplicated code between projects
* Writing unit tests that verify interactions between your project and another project

The Brownie package manager is available from the commandline:

$ brownie pm

**Installing a Package**

Brownie supports package installation from ethPM and Github.

**Installing from Github**

The easiest way to install a package is from a Github repository. Brownie considers a Github repository to be a package if meets the following criteria:

* The repository must have one or more tagged versions.
* The repository must include a contracts/ folder containing one or more Solidity or Vyper source files.

A repository does not have to implement Brownie in order to function as a package. Many popular projects using frameworks such as Truffle or Embark can be added as Brownie packages.

To install a package from Github you must use a package ID. A package ID is comprised of the name of an organization, a repository, and a version tag. Package IDs are not case sensitive.

[ORGANIZATION]/[REPOSITORY]@[VERSION]

It is possible to install from a private Github repository using an API access token like a [personal access token](https://docs.github.com/en/github/authenticating-to-github/creating-a-personal-access-token). This can be provided to Brownie via the GITHUB\_TOKEN environment variable in the form of username:ghp\_token\_secret. See also <https://docs.github.com/en/rest/overview/other-authentication-methods#basic-authentication>.

**Note**

Be careful to avoid exposing your API token in your command history or otherwise, and don’t grant it more permissions than necessary! In this case **repo** permissions should be sufficient.

**Examples**

To install [OpenZeppelin contracts](https://github.com/OpenZeppelin/openzeppelin-contracts) version 3.0.0:

$ brownie pm install OpenZeppelin/openzeppelin-contracts@3.0.0

To install [AragonOS](https://github.com/aragon/aragonOS) version 4.0.0:

$ brownie pm install aragon/aragonos@4.0.0

**Installing from ethPM**

The [Ethereum Package Manager](https://www.ethpm.com/) (ethPM) is a decentralized package manager used to distribute EVM smart contracts and projects.

At its core, an ethPM package is a JSON object containing the ABI, source code, bytecode, deployment data and any other information that combines together to compose the smart contract idea. The [ethPM specification](http://ethpm.github.io/ethpm-spec/) defines a schema to store all of this data in a structured JSON format, enabling quick and efficient transportation of smart contract ideas between tools and frameworks which support the specification.

To obtain an ethPM package, you must know both the package name and the address of the registry where it is available. This information is communicated through a [registry URI](https://docs.ethpm.com/uris#registry-uris). Registry URIs use the following format:

ethpm://[CONTRACT\_ADDRESS]:[CHAIN\_ID]/[PACKAGE\_NAME]@[VERSION]

The Snake Charmers maintain an [ethPM registry explorer](http://explorer.ethpm.com/) where you can obtain registry URIs.

**Examples**

To install OpenZeppelin’s [Math](https://github.com/OpenZeppelin/openzeppelin-contracts/tree/master/contracts/math) package, served from the Snake Charmers [Zeppelin registry](http://explorer.ethpm.com/browse/mainnet/zeppelin.snakecharmers.eth):

$ brownie pm install ethpm://zeppelin.snakecharmers.eth:1/math@1.0.0

To install v2 of the [Compound Protocol](https://github.com/compound-finance/compound-protocol), served from the Snake Charmers [DeFi registry](http://explorer.ethpm.com/browse/mainnet/defi.snakecharmers.eth):

$ brownie pm install ethpm://defi.snakecharmers.eth:1/compound@1.1.0

**Working with Packages**

**Viewing Installed Packages**

Use brownie pm list to view currently installed packages. After installing all of the examples given above, the output looks something like this:

$ brownie pm list

Brownie - Python development framework **for** Ethereum

The following packages are currently installed:

OpenZeppelin

└─OpenZeppelin/openzeppelin-contracts@3.0.0

aragon

└─aragon/aragonOS@4.0.0

zeppelin.snakecharmers.eth

└─zeppelin.snakecharmers.eth/access@1.0.0

defi.snakecharmers.eth

└─defi.snakecharmers.eth/compound@1.1.0

**Cloning a Package**

Use brownie pm clone [path] to copy the contents of a package into another folder. The package will be cloned to the current directory if [path] is ommited. This is useful for exploring the filestructure of a package, or when you wish to build a project on top of an existing package.

To copy the Aragon package to the current folder:

$ brownie pm clone aragon/aragonOS@4.0.0

**Using Packages in your Project**

**Importing Sources from a Package**

You can import sources from an installed package in the same way that you would a source within your project. The root path is based on the name of the package and can be obtained via brownie pm list.

For example, to import SafeMath from OpenZeppelin contracts:

import "OpenZeppelin/openzeppelin-contracts@3.0.0/contracts/math/SafeMath.sol";

You can modify the import path with the remappings field in your project configuration file. See [Remapping Installed Packages](https://eth-brownie.readthedocs.io/en/v1.19.3/compile.html#compile-remap-packages) for more information.

**Using Packages in Tests**

The pm fixture provides access to installed packages during testing. It returns a [**Project**](https://eth-brownie.readthedocs.io/en/v1.19.3/api-project.html#brownie.project.main.Project) object when called with a project ID:

**def** test\_with\_compound\_token(pm):

compound = pm('defi.snakecharmers.eth/compound@1.1.0').CToken

See the [unit test documentation](https://eth-brownie.readthedocs.io/en/v1.19.3/tests-pytest-intro.html#pytest-other-projects) for more detailed information.

**Declaring Project Dependencies**

Dependencies are declared by adding a dependencies field to your project [configuration file](https://eth-brownie.readthedocs.io/en/v1.19.3/config.html#config):

**dependencies**:

- aragon/aragonOS@4.0.0

- defi.snakecharmers.eth/compound@1.1.0

Brownie attempts to install any listed dependencies prior to compiling a project. This is useful when your project may be used outside of your local environment.

**The Brownie GUI**

Brownie includes a GUI for viewing test coverage data and analyzing the compiled bytecode of your contracts.

Parts of this section assume a level of familiarity with EVM bytecode. If you are looking to learn more about the subject, Alejandro Santander from [OpenZeppelin](https://openzeppelin.com/) has written an excellent guide - [Deconstructing a Solidity Contract](https://blog.openzeppelin.com/deconstructing-a-solidity-contract-part-i-introduction-832efd2d7737/).

**Note**

If you receive an error when attempting to load the GUI, you probably do not have Tk installed on your system. See the [Tk installation instrucions](https://eth-brownie.readthedocs.io/en/v1.19.3/install.html#install-tk) for more detailed information.

**Getting Started**

To open the GUI, run the following command from within your project folder:

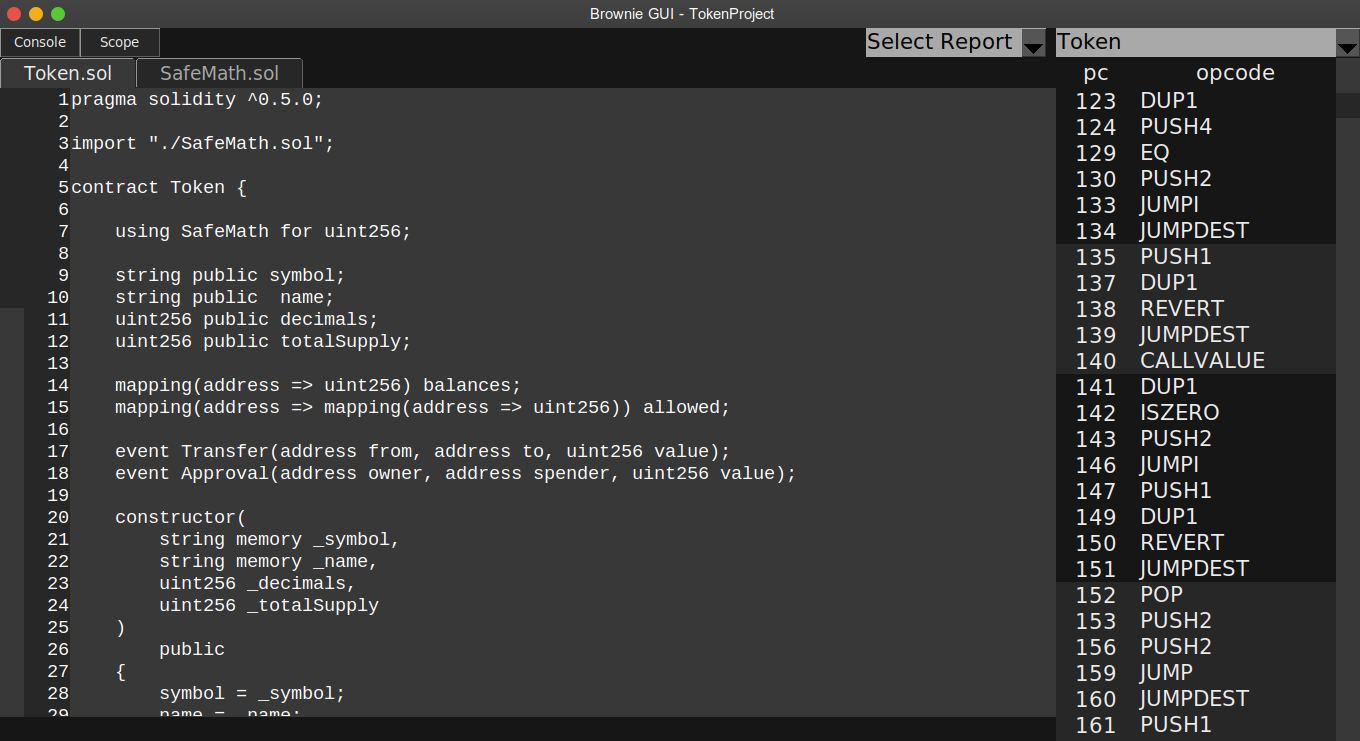
$ brownie gui

Or from the console:

**>>>** Gui()

Once loaded, the first thing you’ll want to do is choose a contract to view. To do this, click on the drop-down list in the upper right that says “Select a Contract”. You will see a list of every deployable contract within your project.

Once selected, the contract source code is displayed in the main window with a list of opcodes and program counters on the right. If the contract inherits from more than one source file, tabs will be available to switch between sources. For example, in the image below the Token contract includes both Token.sol and SafeMath.sol:



**Working with Opcodes**

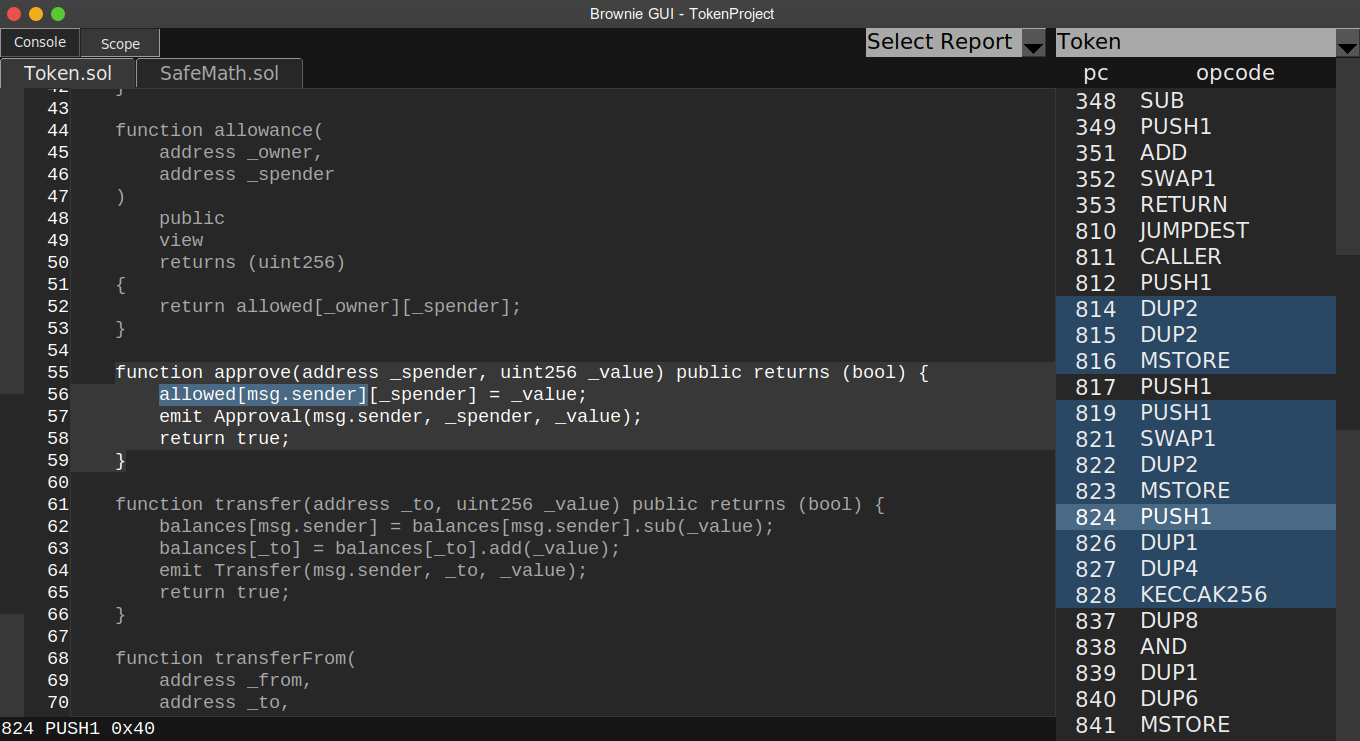
**Mapping Opcodes to Source**

Highlighting a section of code will also highlight the instructions that are associated with it. Similarly, selecting on an instruction will highlight the related source.

Click the Scope button in the top left (or the S key) to filter the list of instructions such that only those contained within the highlighted source are shown.

**Note**

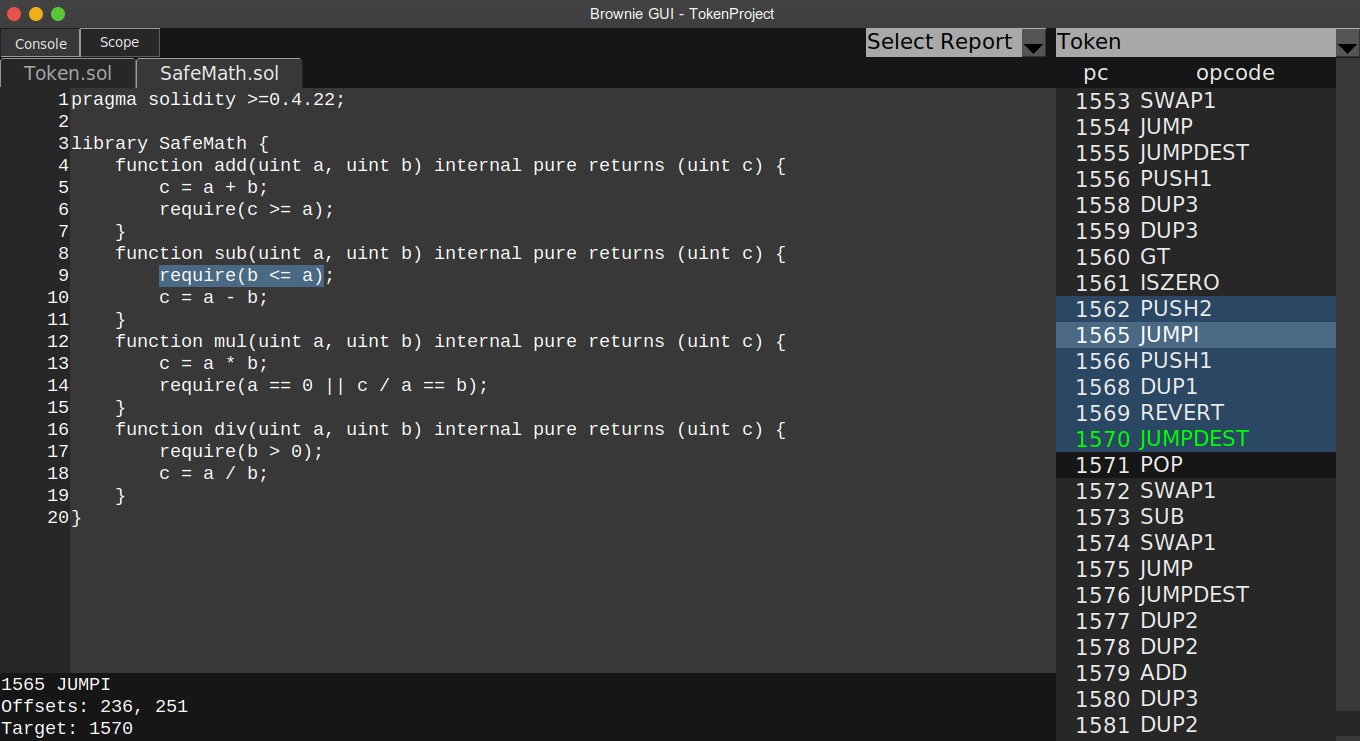
Opcodes displayed with a dark background are not mapped to any source, or are mapped to the source of the entire contract. These are typically the result of compiler optimization or part of the initial function selector.



**Jump Instructions**

Click the Console button in the top left (or press the C key) to expand the console. It shows more detailed information about the highlighted instruction.

* When you select a JUMP or JUMPI instruction, the console includes a “Target:” field that gives the program counter for the related JUMPDEST, where possible. The related JUMPDEST is also highlighted in green. Press the J key to show the instruction.
* When you select a JUMPDEST instruction, the console includes a “Jumps:” field that gives a list of program counters that point at the highlighted instruction. Each related JUMP/JUMPI is also highlighted in green.



**Miscellaneous**

* Right clicking on an instruction will apply a yellow highlight to all instructions of the same opcode type.
* Press the R key to toggle highlight on all REVERT opcodes.

**Viewing Reports**

Actions such as coverage evaluation and security analysis produce report files within the reports/ directory of your project. To examine a report:

1. click on the drop-down list in the upper right that says “Select Report”
2. Select the report file you wish to view.
3. A new drop-down list will appear where you can select which report to display.

Some reports will include additional information that is displayed in the GUI console when you hover the mouse over a related section.

Here is an example of a coverage analysis report:



**Report JSON Format**

Third party tools can generate reports for display in the Brownie GUI. Reports must be saved in the reports/ directory of a project. Brownie expects reports to be JSON encoded and use the following structure:

{

"highlights": {

*// this name is shown in the report type drop-down menu*

"<Report Type>": {

"ContractName": {

"path/to/sourceFile.sol": [

*// start offset, stop offset, color, optional message*

[123, 440, "green", ""],

[502, 510, "red", ""],

]

}

}

},

"sha1": {} *// optional, not yet implemented*

}

The final item in each highlight offset is an optional message to be displayed. If included, the text given here will be shown in the GUI console when the user hovers the mouse over the highlight. To not show a message, set it to "" or null.