Iteration 1 Document: Brownie - A Python-based Development and Testing Framework for Smart Contracts Targeting the Ethereum Virtual Machine

---

Current Focus: Byte Arrays Not Parsed Elegantly (#1568)

We've identified the core issues related to byte array parsing in the Brownie framework. As part of our resolution strategy, we've initiated the development of a package that enhances the capability of the framework in handling various input formats, including hexadecimal.

1. Inputs and Outputs

Inputs:

checkUpkeep function:

tournamentIdBytes (bytes calldata) - Represents the tournament ID in bytes format. It can be any valid bytes data.

Outputs:

checkUpkeep function:

upkeepNeeded (bool) - Returns true if the upkeep is needed, otherwise false.

performData (bytes memory) - Returns the input tournamentIdBytes.

2. Key Data Structures

tournamentIdBytes:

Type: bytes

Description: A bytes representation of the tournament ID. It's decoded to get the actual tournamentId which is then used to determine if upkeep is needed.

tournamentId:

Type: uint

Description: The decoded value of tournamentIdBytes. It's used to check if the ID is even or odd.

3. Use-Cases

Use-Case 1: Testing Various Input Formats

Description:

Brownie should be capable of testing various input formats, such as hexadecimal, not just decimal.

Flow:

Deploy the contract.

Call the checkUpkeep function with various formats of tournamentId, e.g., decimal, hexadecimal.

The function should handle and parse these formats correctly without errors.

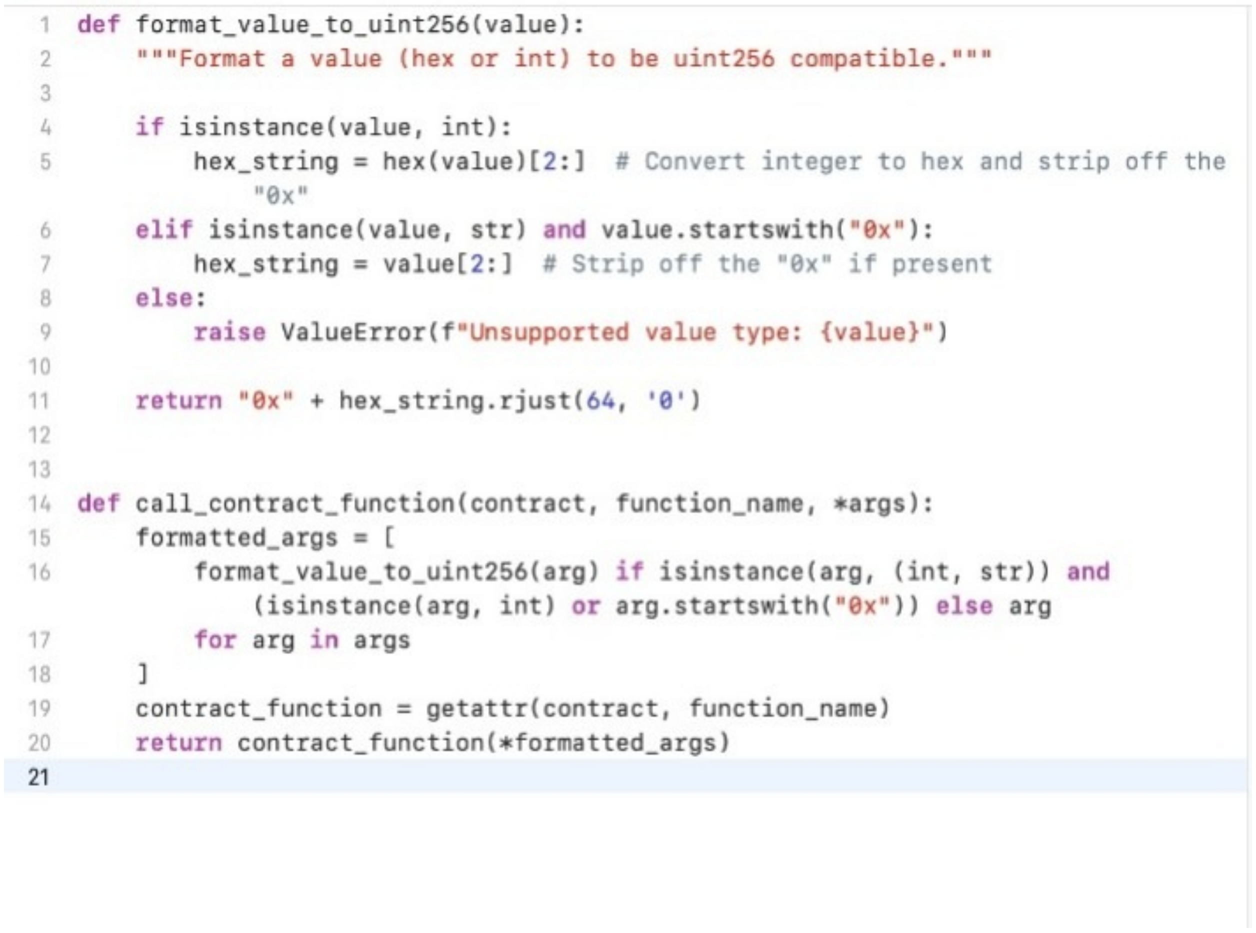
Exceptional Cases:

If an invalid format is provided, the function should return an error or handle it gracefully.

Use-Case 2: Improved Debugging with Logs

Description:

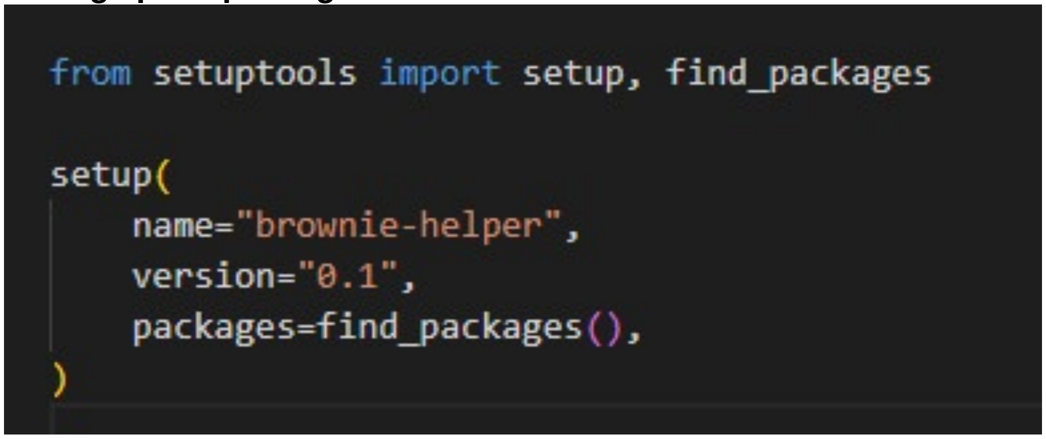
It's challenging to debug without logs. The framework should provide logs to assist in debugging.
Flow:
Deploy the contract.
Call the checkUpkeep function.
If there's an error or exception, the function should log relevant data for debugging.
Exceptional Cases:
If the function runs successfully, no logs related to errors should be generated.
Use-Case 3: Code Breakage
Description:
There are instances where the function breaks the code, and the reason isn't evident.
Flow:
Deploy the contract.
Call the checkUpkeep function.
The function should execute without breaking the code.
Exceptional Cases:
If there's a code breakage, the function should log the cause or provide a detailed error message.



```python
def format_value_to_uint256(value):
    """Format a value (hex or int) to be uint256 compatible."""

    if isinstance(value, int):
        hex_string = hex(value)[2:]  # Convert integer to hex and strip off the "0x"
    elif isinstance(value, str) and value.startswith("0x"):
        hex_string = value[2:]  # Strip off the "0x" if present
    else:
        raise ValueError(f"Unsupported value type: {value}")

    return "0x" + hex_string.rjust(64, '0')


def call_contract_function(contract, function_name, *args):
    formatted_args = [
        format_value_to_uint256(arg) if isinstance(arg, (int, str)) and (isinstance(arg, int) or arg.startswith("0x")) else arg
        for arg in args
    ]
    contract_function = getattr(contract, function_name)
    return contract_function(*formatted_args)

```

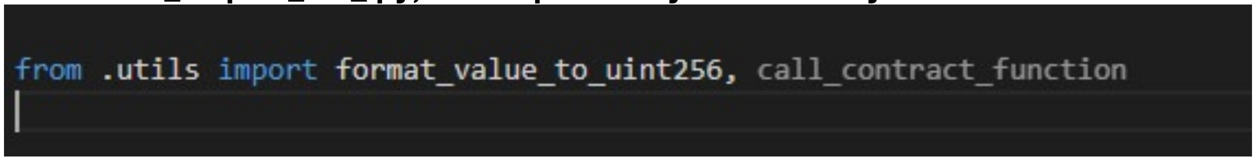
this is the package we created that can handle hexadecimal and decimal integer along with exception with proper errors to help debugging the problem.

setting up the package:



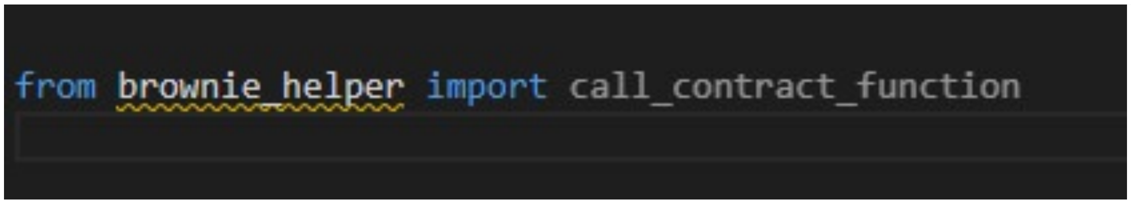
```
from setuptools import setup, find_packages

setup(
    name="brownie-helper",
    version="0.1",
    packages=find_packages(),
)
```

In brownie_helper/_init_.py, add imports so you can easily use the functions:



```
from .utils import format_value_to_uint256, call_contract_function
```

now we can use this package to solve this problem and also publish this for anyone to solve this problem:



```
from brownie_helper import call_contract_function
```

Customers and Users:
1. Smart Contract Developers - Requirements: Intuitive development environment, debugging tools, and Ethereum network integration.
2. Dapp Developers- Requirements: Strong framework for contract interactions, efficient deployment mechanisms, and insight into transactions.
3. Auditors and Security Researchers - Requirements: Extensive testing, simulation capabilities, and insight into contract behavior.
4. Educators and student - Requirements: User-friendly interface, documentation, and a supportive community.
Feedback Channel:
- GitHub Issues:*Direct insights from users.
- Online forums and chats: Platforms like Ethereum Stack Exchange and Discord.
- Workshops & Meetups: Direct user interaction.

Documentation:
- Tutorials and Guides: Helps users understand the features of Brownie.
- Feedback Log: Tracks user feedback chronologically.

In the upcoming iteration, our primary focus will be on addressing the issue related to Error with Vyper Tests (#1653). This critical issue has been identified as a top priority, and we are committed to resolving it to ensure seamless Vyper test execution within the Brownie tool. Our team will conduct a thorough investigation, implement necessary changes, and rigorously test the solution to provide a reliable and improved experience for our users

---

References:

[1] Brownie GitHub Repository. Issues #1653 and #1568. Available at: [GitHub Link] [2] Truffle Suite. Available at: [Truffle Official Website] [3] Remix Ethereum IDE. Available at: [Remix Official Website] [4] Hardhat Developer Environment. Available at: [Hardhat Official Website] [5] Brownie Documentation. Dependency Management. Available at: [Brownie Docs Link] [6] Brownie Project Mentorship Program. Available at: [Mentorship Program Link] [7] Brownie User Community Feedback. Available at: [Community Feedback Link]

Note: All third-party material, visuals, text, and code have been fully cited in accordance with best practices