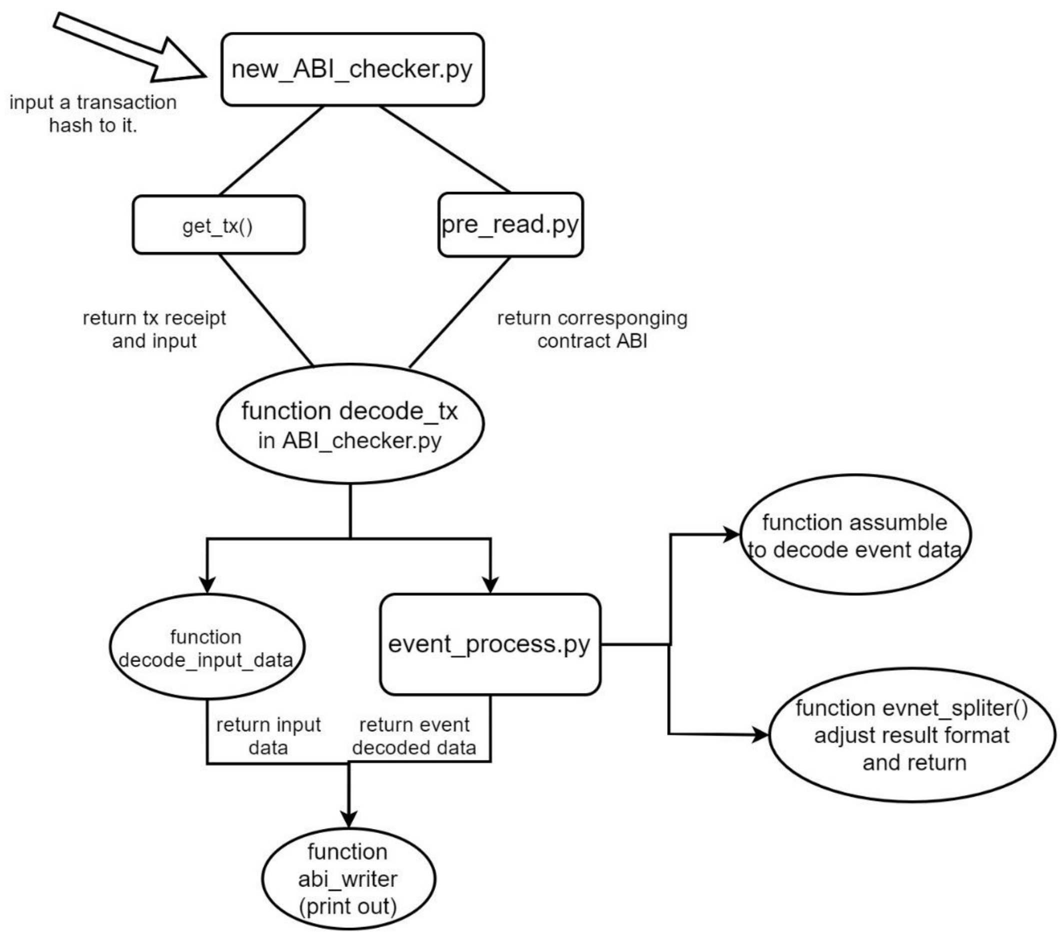
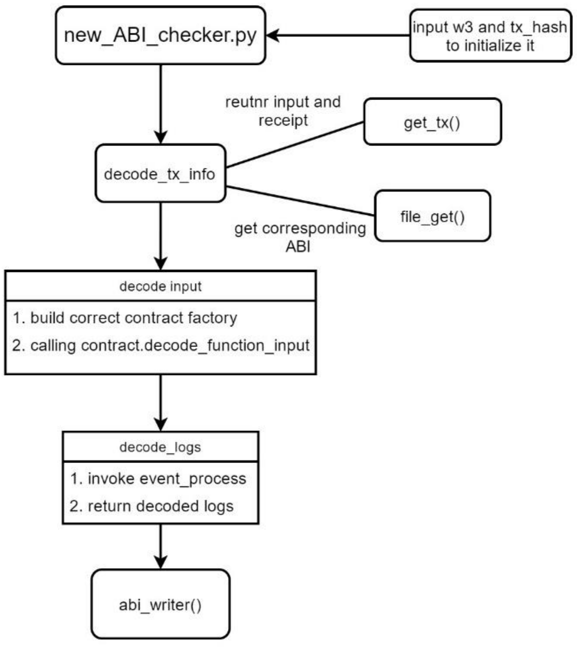
A brief introduction of my codes:



input a transaction hash to it.
new_ABI_checker.py
get_tx()
pre_read.py
return tx receipt and input
return corresponging contract ABI
function decode_tx in ABI_checker.py
function decode_input_data
event_process.py
function assumble to decode event data
function evnet_spliter() adjust result format and return
return input data
return event decoded data
function abi_writer (print out)

A detailed introduction of new_Abi_checker.py



Mainly it has three functions:

1. def file_get(self, Address : string).

This function used to detect whether there is corresponding contract in local folder and read it. If there isn't, it will download the contract from etherscan by address input["to"]

2. def get_tx(self)

Return input and receipt

3. def decode_tx_info(self)

This function is used to decode tx input and logs, usually it will process input first.

Input is processed by web3 function named

    Contract.decode_function_input(input[data])

Before using this function, you need to build a Contract factory,

    Contract = web3.eth.contract(contract_address, ABI)

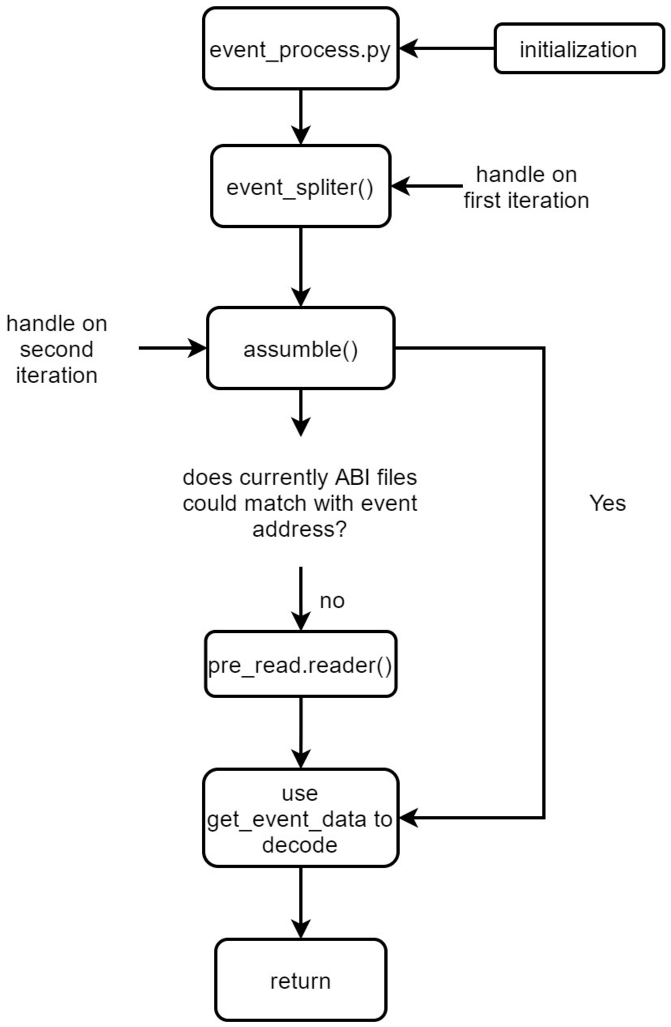
Usually decode_function_input will return an tuple which contain the input function full name and corresponding decoded data.

from line 81 to 95, we will insert decoded data into our specified dict "matched" in line 102 logs are processed by class event_checker. Then it will return another dict which stores the decoded data of a logs.

In the last the two dict would be put into a list and send to abi_writer() to convert them into json file.

Tips: you can just ignore any syntax about object Otimes, it only used to calculate the running time

A detailed introduction of event_process.py



event_process.py
initialization
event_spliter()
handle on first iteration
handle on second iteration
assumble()
does currently ABI files could match with event address?
Yes
no
pre_read.reader()
use get_event_data to decode
return

def event_spliter(self)

the outcome of a logs usually has two-tier, which means it need two iterations to access into value of a logs. Beyond is the two-layer structure of a log.

First layer:

```
"blockHash":"0xa6c5775e9c6cde6b3d9de2230d222eb03ade80798d6ebac3e
3a4e0ddfb7cd817",
"blockNumber": 14214094,
"contractAddress": null,
"cumulativeGasUsed": 4591758,
"effectiveGasPrice": 138624453674,
"from": "0xaEB2584fD2C1d1C27dC72afcb8e858a5fFE4C794",
"gasUsed": 1435133,
"logs": [….]
```

Usually, the value logs matched is a list, inside the list there are many dictionary data. data structure of the dict is like:

```
{
    "anonymous": false,
    "inputs": [
            {
                "indexed": true,
                "internalType": "address",
                "name": "usr",
                "type": "address",
                "result":
"0x3c1ac43BB661246B8218b88D90c902331c58CCDD"
            },
            {
                "indexed": true,
                "internalType": "address",
                "name": "own",
                "type": "address",
                "result":
"0x3c1ac43BB661246B8218b88D90c902331c58CCDD"
            },
            {
                "indexed": true,
                "internalType": "uint256",
                "name": "cdp",
                "type": "uint256",
                "result": 27542
            }
       ],
       "name": "NewCdp",
       "type": "event",
       "signature": “” ,
       "logIndex": 55,
       "transactionIndex": 78,
       "transactionHash": “”,
       "address":"0x5ef30b9986345249bc32d8928B7ee64DE9435E39
",
       "blockHash": "",
       "blockNumber": 14214094
  }
```

For aesthetic here we delete some data like value of signature, transactioinHash

So, in fact, the utility of event_spliter is to access the first iteration, the function assumble actually handles each event

def assumble(self, logs : Dict)

because the event usually has/interacts with its own contract (completely independent with transaction interacted contract). Code will first check its source code existence, if not, we then download ABI from Etherscan.

In line 52, I try to merge two different dict together is to reduce redundant file reads.
Actually, this part could be improved a lot to increase the running speed.
1. if self.event could always exist during runtime, or there is a special ABI standard library that allows us to directly get needed ABI through signature.
2. if (1) can be achieved, then we can store the ABI files name that the computer has read, therefore time for re-reading the file again could be solved
But there is one vital problem: every time when class event_process finished its work and go to the next stage, the whole class would be destroyed from memory to alleviate the workload. Under this situation, how to maintain the value of self.event is a huge obstacle.

Considering that decoding input and logs do not require the same file flow or memory space, we can set this two-stage running concurrently for a faster running time.