*1. Listening for contract events on EVM chains*

# Overview

This is the 4th of 5 assignments in the Bridge Project.

In this assignment, you will be using the web3 library to scan the blockchain for events emitted by contracts on the blockchain.
Unlike the previous two assignments, in this assignment, you will *not* be writing any smart contract code.

# Connecting to the blockchain(s)

In this assignment, you will be **reading** data from the Avalanche and/or BSC testnets. Both networks provide free RPC endpoints that you can use.
As in the first assignment, you can use the following RPC endpoints

The Avalanche “Fuji” C-Chain testnet can be reached:
https://api.avax-test.network/ext/bc/C/RPC

The BSC testnet can be reached:
https://data-seed-prebsc-1-s1.binance.org:8545/

# Filtering events

Ethereum nodes provide a way to “filter” transactions based on certain events, using the web3 library, you can do this using the createFilter command.
The code snippet below shows how to get all the “Transfer” events from the USDC contract on the Ethereum mainnet.

```
from web3 import Web3
from web3.contract import Contract
```

## 1. Listening for contract events on EVM chains

```
eth_usdc_address = Web3.to_checksum_address("0xa0b86991c6218b36c1d19d4a
TRANSFER_ABI = json.loads('[{"name": "Transfer", "inputs": [{"name": "s

api_url = f"PUT YOUR ETH RPC ENDPOINT HERE" #Ethereum mainnet
w3 = Web3(Web3.HTTPProvider(api_url))

contract = w3.eth.contract(address=eth_usdc_address, abi=TRANSFER_ABI)

arg_filter = {}

end_block = w3.eth.get_block_number()
start_block = end_block - 5

event_filter = contract.events.Transfer.createFilter(fromBlock=start_bl
events = event_filter.get_all_entries()

for evt in events:
    data = {
        'to': evt.args['receiver'],
        'from': evt.args['sender'],
        'value': evt.args['value'],
        'transactionHash': evt.transactionHash.hex(),
        'address': evt.address,
    }
    print( json.dumps( data, indent=2 ) )
```

The values, "receiver", "sender", “value” that are returned in

If you use a slightly different ABI, then you must use different evt.args.

The web3 library does **not** check if the ABI you provided is correct, or complete. For example, in the code snippet above, the
ABI we provided (TRANSFER_ABI), is not the full ABI for the

*1. Listening for contract events on EVM chains*

This is sufficient for our example, since that is the only event we are filtering on. If you wanted to get other ERC20 events (e.g. “Approve”), you would need to include an expanded ABI.

createFilter allows you to specify a block range (fromBlock,toBlock), so you might think that if you wanted to get *all* the events from a specific contract,
you could specify fromBlock = 1, and toBlock to be the latest block (e.g. ~19M). Unfortunately, if you specify a moderately large block range your request will
almost definitely time out. For high-traffic contracts, even a block range of a few hundred blocks will often time out. If you want to get events from a large
range of blocks, you’ll need to wrap the createFilter call in a for loop. For the purposes of this assignment, since we are dealing with fairly small block ranges,
and contracts that are not emitting a lot of events you do **not** need to use a for loop, you can simply use fromBlock and toBlock directly.

# Assignment

Using the skeleton listener.py, complete the function scanBlocks().

scanBlocks takes a chain (either ‘avax’ or ‘bsc’), a range of blocks and a contract address,
and scans the chain for ‘Deposit’ events emitted by the contract at the specified address.

More specifically scanBlocks takes 4 arguments

## 1. Listening for contract events on EVM chains

- start_block - the block number of the first block to scan
- end_block - the block number of the last block to scan
- contract_address - the on-chain address of the deposit contract

Deposit events log three values, "token", “to” and "amount". You should record these events along with the
chain (either ‘avax’ or ‘bsc’), the contract address and the transaction hash.

You should record all the events you see to the file "deposit_logs.csv".

Your deposit_logs file should have (at least) the following 6 columns

- chain - String (either ‘avax’ or ‘bsc’)
- token - Address of deposit token (in hex)
- recipient - Address of recipient (in hex)
- amount - Number of tokens being transferred
- transactionHash - transaction hash (in hex)
- address - The address of the contract that emitted the event (in hex)

The grader uses the Pandas CSV library to read your deposit_logs file, and expects the file to be formatted a specific way. We encourage you to use the pandas CSV writer to create and populate your deposit_logs.csv. However you decide to create it the file must contain, on the first line, a comma separated “headers” list followed by as many rows of comma separated data as you need. In the following example we have abbreviated the hex string to make it easier to read

```
chain,token,recipient,amount,transactionHash,address,date
avax,0xc<hex omitted>,0x6<hex omitted>,466,0x5<hex omitted>,0x2<hex omi
```

# 1. Listening for contract events on EVM chains

other events and transactions with the contract

In a real bridge, your function scanBlocks() would probably run as a background process, scanning every new block as it is produced.
For the purposes of this assignment, scanBlocks will be run on demand by the autograder, and you may assume that the autograder provides the correct
range of blocks.

Next ▶