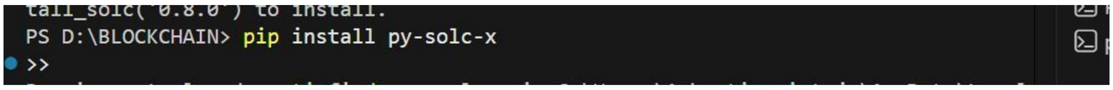
ASSIGNMENT - 004

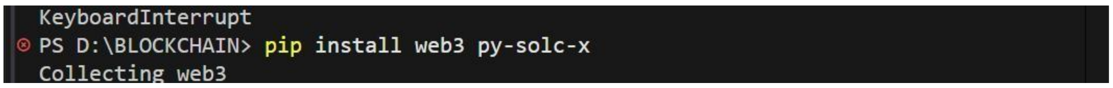
2303A51547
S.SRIRAM
BATCH-25

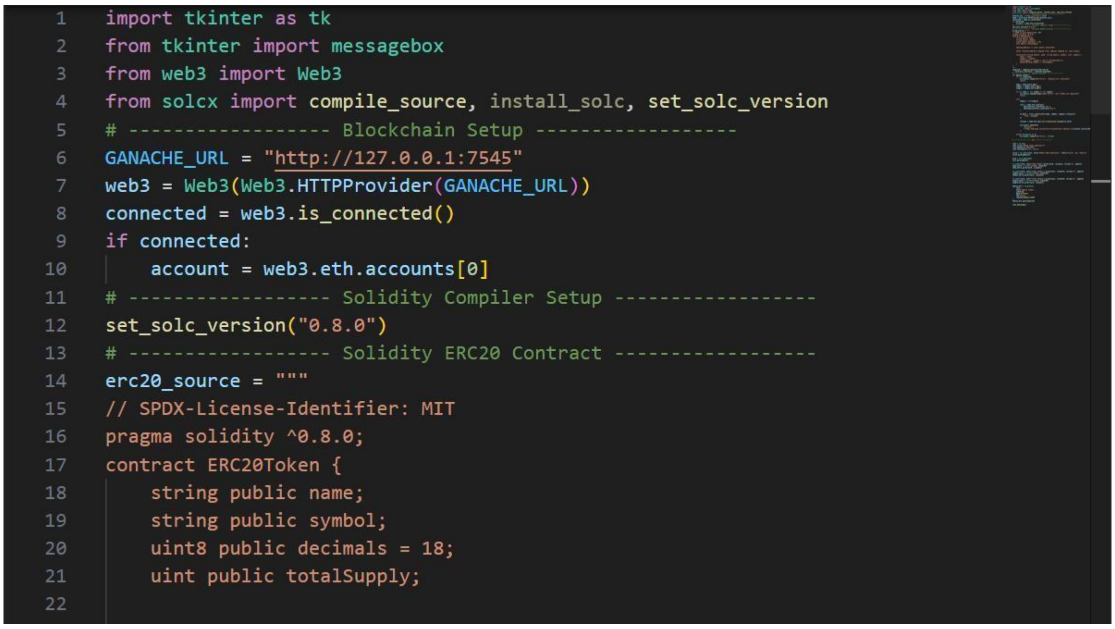
Objective
To design, implement, and deploy a basic ERC-20 compliant token smart contract using Solidity on the Ethereum blockchain.
Requirements
- Install VS Code
- Install Solidity extension in VS Code
- Use Remix Ethereum IDE (online) or Hardhat (optional)
- Basic understanding of blockchain and Ethereum
- MetaMask wallet (for deployment testing)

Practical Implementation
Step 1: Development Environment Setup
- Install VS Code

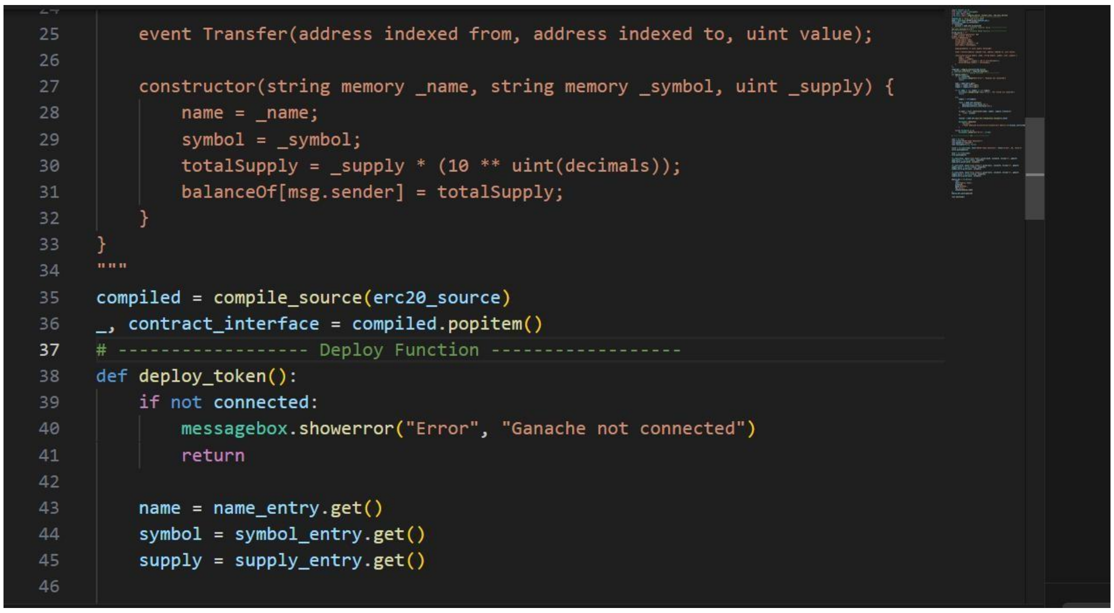
Step 2: Create ERC20 Smart Contract
Create a Solidity smart contract that:
- Defines token name, symbol, decimals, and total supply
- Allows token transfer between accounts
- Maintains balances using mappings
- Emits events for transparency

Code:

```
tall_solc('0.8.0') to install.
PS D:\BLOCKCHAIN> pip install py-solc-x
>>
```

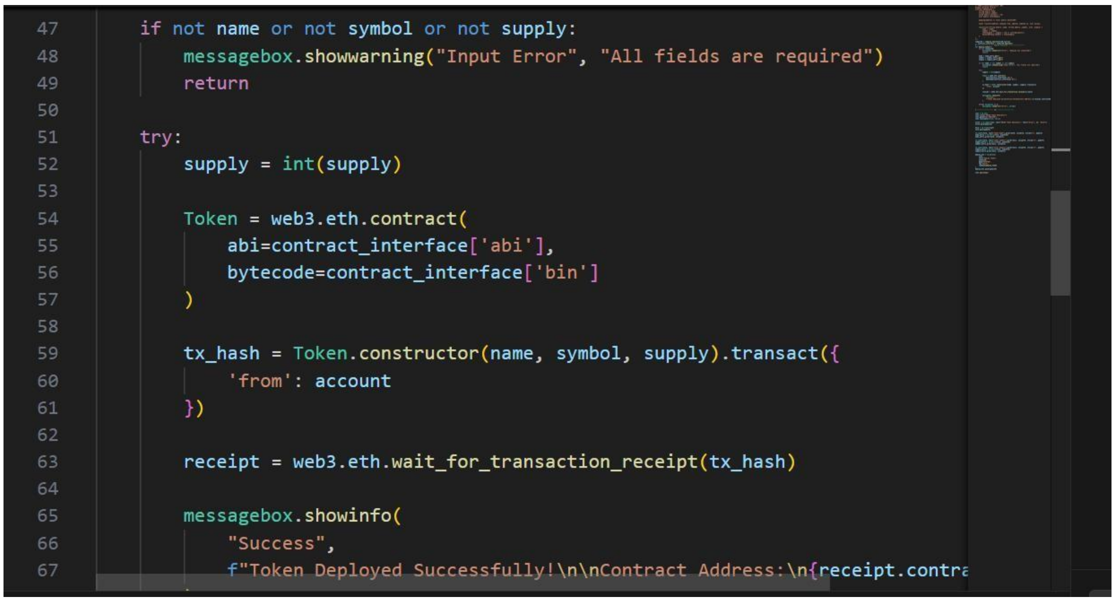

```
KeyboardInterrupt
PS D:\BLOCKCHAIN> pip install web3 py-solc-x
Collecting web3
```



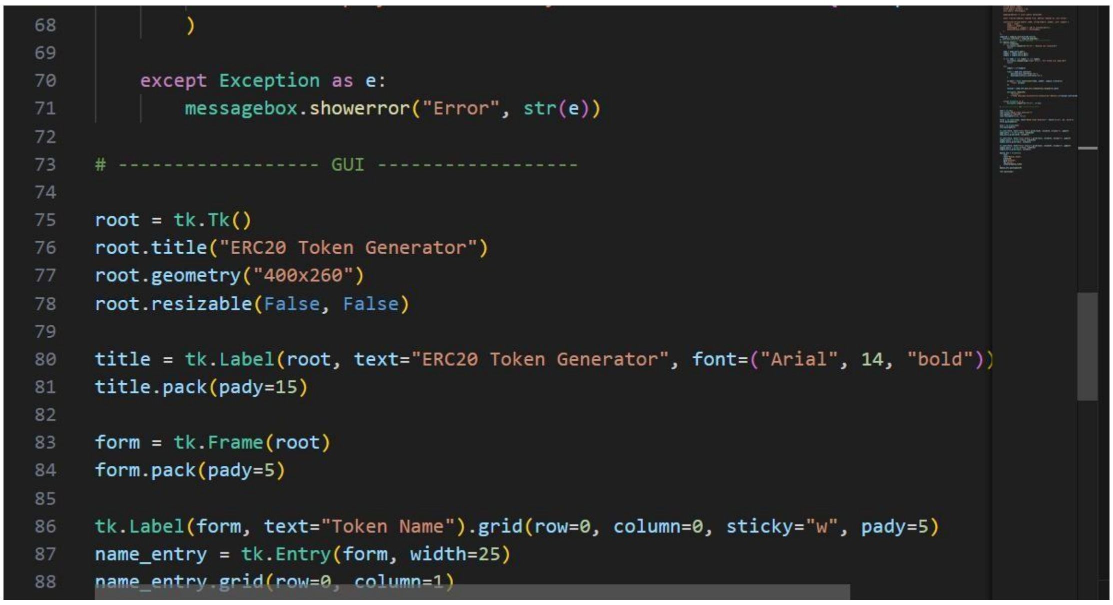
```python
import tkinter as tk
from tkinter import messagebox
from web3 import Web3
from solcx import compile_source, install_solc, set_solc_version
# ------------------ Blockchain Setup ------------------
GANACHE_URL = "http://127.0.0.1:7545"
web3 = Web3(Web3.HTTPProvider(GANACHE_URL))
connected = web3.is_connected()
if connected:
    account = web3.eth.accounts[0]
# ------------------ Solidity Compiler Setup ------------------
set_solc_version("0.8.0")
# ------------------ Solidity ERC20 Contract ------------------
erc20_source = """
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract ERC20Token {
    string public name;
    string public symbol;
    uint8 public decimals = 18;
    uint public totalSupply;

```



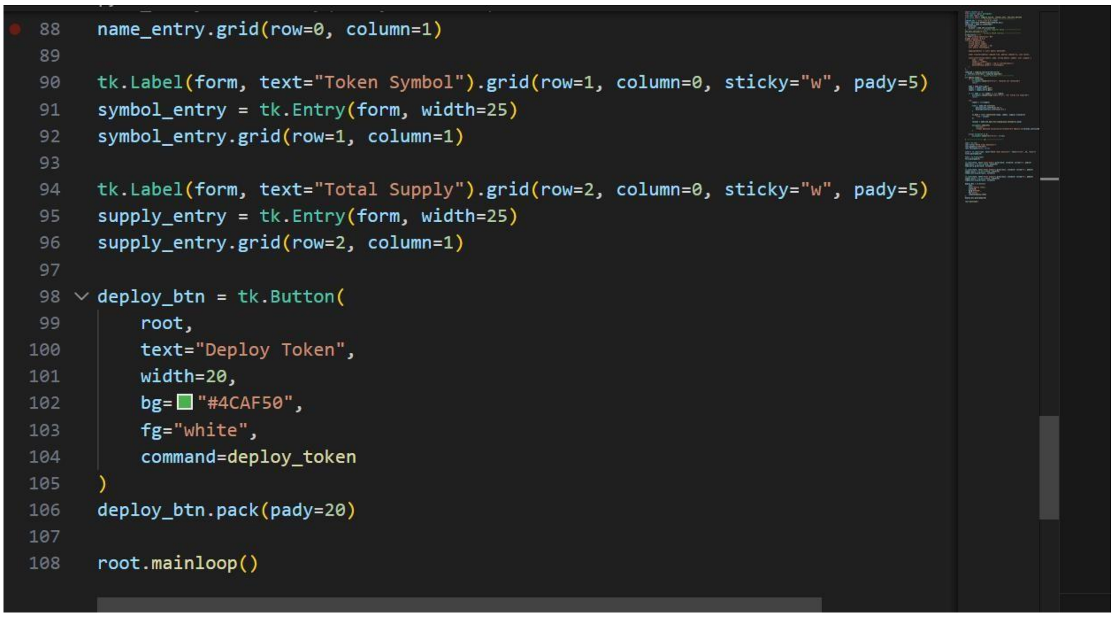
```python
    event Transfer(address indexed from, address indexed to, uint value);

    constructor(string memory _name, string memory _symbol, uint _supply) {
        name = _name;
        symbol = _symbol;
        totalSupply = _supply * (10 ** uint(decimals));
        balanceOf[msg.sender] = totalSupply;
    }
}
"""
compiled = compile_source(erc20_source)
_, contract_interface = compiled.popitem()
# ------------------ Deploy Function ------------------
def deploy_token():
    if not connected:
        messagebox.showerror("Error", "Ganache not connected")
        return

    name = name_entry.get()
    symbol = symbol_entry.get()
    supply = supply_entry.get()

```



```python
    if not name or not symbol or not supply:
        messagebox.showwarning("Input Error", "All fields are required")
        return

    try:
        supply = int(supply)

        Token = web3.eth.contract(
            abi=contract_interface['abi'],
            bytecode=contract_interface['bin']
        )

        tx_hash = Token.constructor(name, symbol, supply).transact({
            'from': account
        })

        receipt = web3.eth.wait_for_transaction_receipt(tx_hash)

        messagebox.showinfo(
            "Success",
            f"Token Deployed Successfully!\n\nContract Address:\n{receipt.contra
```

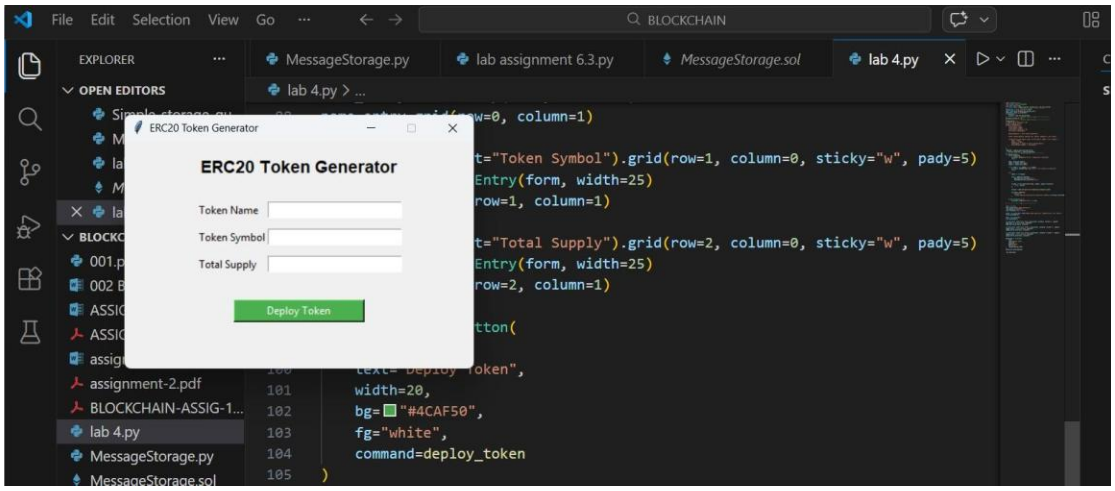


```python
        )

    except Exception as e:
        messagebox.showerror("Error", str(e))

# ----------------- GUI -----------------

root = tk.Tk()
root.title("ERC20 Token Generator")
root.geometry("400x260")
root.resizable(False, False)

title = tk.Label(root, text="ERC20 Token Generator", font=("Arial", 14, "bold"))
title.pack(pady=15)

form = tk.Frame(root)
form.pack(pady=5)

tk.Label(form, text="Token Name").grid(row=0, column=0, sticky="w", pady=5)
name_entry = tk.Entry(form, width=25)
name_entry.grid(row=0, column=1)
```



```python
name_entry.grid(row=0, column=1)

tk.Label(form, text="Token Symbol").grid(row=1, column=0, sticky="w", pady=5)
symbol_entry = tk.Entry(form, width=25)
symbol_entry.grid(row=1, column=1)

tk.Label(form, text="Total Supply").grid(row=2, column=0, sticky="w", pady=5)
supply_entry = tk.Entry(form, width=25)
supply_entry.grid(row=2, column=1)

deploy_btn = tk.Button(
    root,
    text="Deploy Token",
    width=20,
    bg=🟩"#4CAF50",
    fg="white",
    command=deploy_token
)
deploy_btn.pack(pady=20)

root.mainloop()
```

Output:



ERC20 Token Generator

```
w=0, column=1)

t="Token Symbol").grid(row=1, column=0, sticky="w", pady=5)
Entry(form, width=25)
row=1, column=1)

t="Total Supply").grid(row=2, column=0, sticky="w", pady=5)
Entry(form, width=25)
row=2, column=1)

tton(

    oken",
    width=20,
    bg="#4CAF50",
    fg="white",
    command=deploy_token
)
```

- State variables = name, symbol, decimals, totalSupply, balanceOf
- Constructor = constructor(uint initialSupply)
- Token transfer function = transfer(address to, uint value)
- Balance storage = mapping(address => uint) balanceOf
- Event used = Transfer(address from, address to, uint value)