Problem 1: My First Cryptocurrency (ERC20)

Objective Learn the fundamentals of the ERC20 standard by creating your own basic, tradable cryptocurrency on the blockchain.

Learn These Elements First

- `pragma solidity ^0.8.20;`
    - **What it is:** The very first line in any Solidity file. It tells the compiler which version of the language to use, preventing issues with future, breaking changes. `^0.8.20` means "use version 0.8.20 or any newer version in the 0.8.x series."
- `import "@openzeppelin/contracts/...";`
    - **What it is:** Solidity allows you to import code from other files. OpenZeppelin is a library of secure, community-vetted smart contracts. By importing their `ERC20.sol` contract, you get all the standard token functionality without having to write it from scratch, which is both easier and safer.
- `contract MyToken is ERC20 { ... }`
    - **What it is:** This declares a new contract named `MyToken`. The `is ERC20` part means your contract *inherits* all the features and functions from the imported OpenZeppelin ERC20 contract.
- `constructor(...) ERC20(...) { ... }`
    - **What it is:** The `constructor` is a special function that runs only **once** when the contract is first deployed. Here, we use it to set the token's name and symbol. The `ERC20(name, symbol)` part passes this information up to the parent OpenZeppelin contract.
- `_mint(address to, uint256 amount)`
    - **What it is:** This is the function that creates new tokens. `_mint` is an "internal" function from the imported ERC20 contract. You call it inside your constructor to create the initial supply of your token and assign it to a specific address.

Your Task

You must develop an ERC20-compliant smart contract that creates a new cryptocurrency.

- The token collection shall be named **"Pixel Token"** with the symbol **"PXL"**.
- The contract's deployment must trigger the creation of an initial total supply of **1,000,000** tokens.
- The entirety of this initial supply must be allocated to the wallet address that deploys the contract.

Evaluation Criteria:

1. **ERC20 Compliance**: Does the token correctly implement the ERC20 standard?
2. **Supply Management**: Is the initial supply correctly set and assigned to the deployer upon construction?
3. **Code Quality**: Is the code clean, secure, and efficient?

Problem 2: Verifiable Digital Certificates (NFTs)

Objective Learn the ERC721 standard for non-fungible tokens (NFTs) by creating a system that issues unique, verifiable certificates of completion.

Learn These Elements First

- **ERC721 Standard**
    - **What it is:** The standard for non-fungible tokens. Unlike ERC20 where all tokens are identical, every ERC721 token is unique and has a specific `tokenId`. This is perfect for representing ownership of one-of-a-kind items like artwork, collectibles, or certificates.
- `struct`
    - **What it is:** A way to create your own custom data type by grouping several variables together. It's perfect for defining the properties of your certificate.
    - *Example:* `struct Certificate { string courseName; uint256 completionDate; }`
- `_safeMint(address to, uint256 tokenId)`
    - **What it is:** The standard function from OpenZeppelin's `ERC721` contract for creating a new, unique NFT and assigning it to a specific address. The `tokenId` must be a number that has not been used before.
- **`onlyOwner` Modifier**
    - **What it is:** A piece of reusable code (a `modifier`) provided by OpenZeppelin's `Ownable` contract. By adding `onlyOwner` to a function definition, you restrict its access so that only the address that deployed the contract can call it. This is essential for controlling who can issue certificates.
- `tokenId`
    - **What it is:** The unique identifier for each NFT. The first NFT you mint is `tokenId` 0, the next is 1, and so on. You need a counter in your contract to keep track of the next available ID.

Your Task

You must develop an ERC721-compliant smart contract named **`CertificateIssuer`** that can issue unique, non-fungible "Proof of Completion" certificates.

- The NFT collection shall be named **"Proof of Completion"** with the symbol **"POC"**.
- The contract must expose a function, restricted to the contract owner, that can mint a new certificate NFT and assign it to a specified recipient address.
- Crucially, the contract must provide a mechanism to permanently associate custom data with each certificate NFT. This data must include the recipient's name and the title of the course they completed. This association must be stored on-chain and be publicly readable.

Evaluation Criteria:

1. **Functionality**: Does the contract correctly mint unique NFTs to the specified recipients?
2. **Access Control**: Is the minting function properly restricted to the contract owner?
3. **On-Chain Data**: Is the certificate data (name, course) correctly stored on-chain and linked to the appropriate token ID?
4. **Code Quality**: Is the code clean, well-structured, and secure?