1. Finding an Input String for a SHA-256 Hash Starting with a Specific Prefix

```
import hashlib

def find_hash_with_prefix(prefix, max_nonce=1000000):
    for nonce in range(max_nonce):
        text = f"input{nonce}".encode()
        hash_result = hashlib.sha256(text).hexdigest()
        if hash_result.startswith(prefix):
            return text.decode(), hash_result
    return None, None

prefix = "00000"
result, hash_value = find_hash_with_prefix(prefix)
print(f"Input: {result}")
print(f"Hash: {hash_value}")
```

2. Finding an Input String That Starts with a Specific Prefix and a Specific Hash Prefix

```
import hashlib

def find_hash_with_prefix_and_input(start_prefix, hash_prefix, max_nonce=1000000):
    for nonce in range(max_nonce):
        text = f"{start_prefix}{nonce}".encode()
        hash_result = hashlib.sha256(text).hexdigest()
        if hash_result.startswith(hash_prefix):
            return text.decode(), hash_result
    return None, None
```

```
start_prefix = "devadnani26"
hash_prefix = "00000"
result, hash_value = find_hash_with_prefix_and_input(start_prefix, hash_prefix)
print(f"Input: {result}")
print(f"Hash: {hash_value}")
```

3. Finding a Nonce for a Specific Input to Get a Hash with a Certain Prefix

```
const crypto = require('crypto');

// Function to find a nonce for a specific input string with a desired hash prefix
function findNonceForInput(inputString, hashPrefix, maxNonce = 1000000) {
  for (let nonce = 0; nonce < maxNonce; nonce++) {
    const text = `${inputString}${nonce}`;
    const hash = crypto.createHash('sha256').update(text).digest('hex');
    if (hash.startsWith(hashPrefix)) {
      return { nonce, hash };
    }
  }
  return { nonce: null, hash: null };
}

const inputString = "Dev => Karan | Rs 100\nKaran => Darsh | Rs 10";
const hashPrefix = "00000";
const result = findNonceForInput(inputString, hashPrefix);

if (result.nonce !== null) {
  console.log(`Nonce: ${result.nonce}`);
```

```
  console.log(`Hash: ${result.hash}`);
} else {
  console.log("No valid nonce found within the range.");
}
```

5: Donation Smart Contract: Allows individuals to donate or transfer money from their personal accounts to various recipients.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Donation {
  address public owner;

  // Event to log donations
  event DonationReceived(address indexed donor, uint amount);

  constructor() {
    owner = msg.sender;
  }

  // Modifier to restrict actions to the owner
  modifier onlyOwner() {
    require(msg.sender == owner, "Only the owner can perform this action");
    _;
  }

  // Function to donate to the contract
```

```
    function donate() public payable {
        require(msg.value > 0, "Donation must be greater than zero");
        emit DonationReceived(msg.sender, msg.value);
    }

    // Function to withdraw funds by the owner
    function withdraw(uint amount) public onlyOwner {
        require(address(this).balance >= amount, "Insufficient balance");
        payable(owner).transfer(amount);
    }

    // Function to check the contract's balance
    function getBalance() public view returns (uint) {
        return address(this).balance;
    }
}
```

Q:-4:-