Hands-on Activity 2.1: The Tower of Hanoi Problem

Code

```python
# pegs and disks
A = [10, 8, 5, 3]
B = []
C = []

# function to solve tower of hanoi
def hanoiSol(n, src, aux, dest):
    # Base case
    if n == 1:
        disk = src.pop()
        dest.append(disk)
        print(f"Move disk {disk} from A to C")
    else:
        # move n-1 disks from A to B
        hanoiSol(n - 1, src, dest, aux)

        # move the largest disk from A to C
        disk = src.pop()
        dest.append(disk)
        print(f"Move disk {disk} from A to C")

        # move n-1 disks from B to C
        hanoiSol(n - 1, aux, src, dest)

# driver code
print("Steps to solve Tower of Hanoi:")
hanoiSol(len(A), A, B, C)
print(f"\nFinal state: \nA: {A} \nB: {B} \nC: {C}")
```

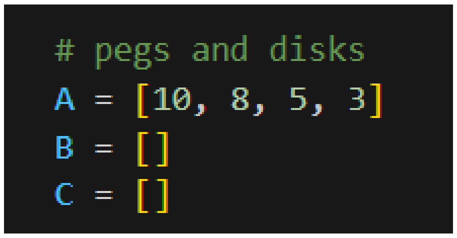
```
Steps to solve Tower of Hanoi:
Move disk 3 from A to C
Move disk 5 from A to C
Move disk 3 from A to C
Move disk 8 from A to C
Move disk 3 from A to C
Move disk 5 from A to C
Move disk 3 from A to C
Move disk 10 from A to C
Move disk 3 from A to C
Move disk 5 from A to C
Move disk 3 from A to C
Move disk 8 from A to C
Move disk 3 from A to C
Move disk 5 from A to C
Move disk 3 from A to C
```

```
Final state:
A: []
B: []
C: [10, 8, 5, 3]
```

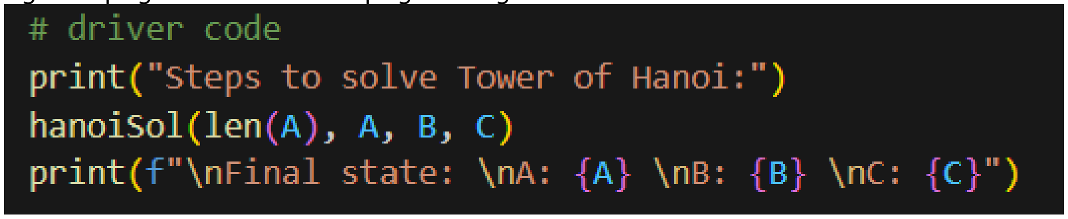
Conclusion

The activity focus on creating a program that solves the Tower of Hanoi problem. It highlighted the use of recursion and dynamic programming, which involves using logical and algorithmic thinking in formulating a solution to the problem. The solution I made particularly involved the recursion method, where the function called itself over and over again until the if condition was followed.

To accomplish this, I created a function named `hanoiSol`. This function takes four parameters: the number of disks or n, the source peg or src, the auxiliary peg or aux, and the destination peg or dest. The function uses an if-else statement to determine whether to execute the base case or continue with the recursive calls. I then declared three lists representing the three pegs (A, B, and C) and initialized peg A with four disks of different values. I also ran the function with the length of peg A and the three pegs as arguments.

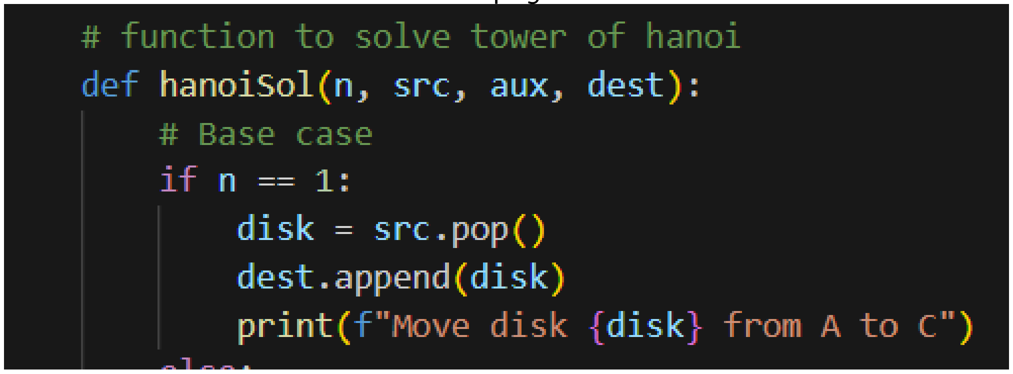


```python
# pegs and disks
A = [10, 8, 5, 3]
B = []
C = []
```



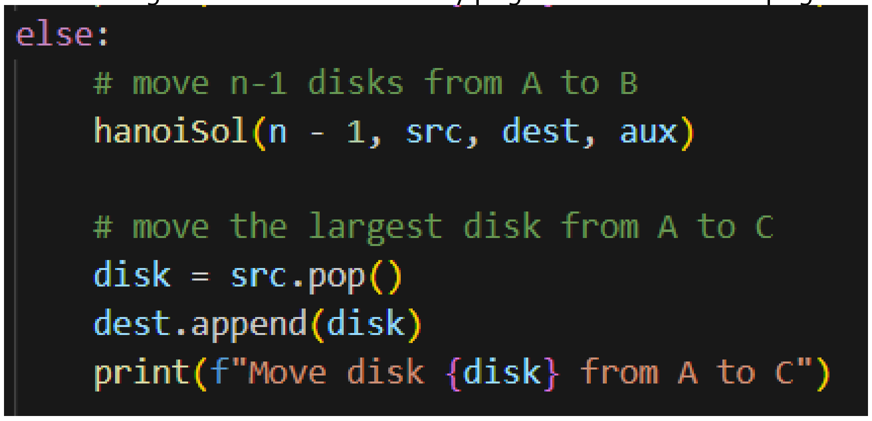
```python
# driver code
print("Steps to solve Tower of Hanoi:")
hanoiSol(len(A), A, B, C)
print(f"\nFinal state: \nA: {A} \nB: {B} \nC: {C}")
```

The if statement says that the disk number must be equal to 1. This represents the stopping condition of the first step of the Tower of Hanoi problem, where the source peg must only have one disk left to move to the destination peg.



```python
# function to solve tower of hanoi
def hanoiSol(n, src, aux, dest):
    # Base case
    if n == 1:
        disk = src.pop()
        dest.append(disk)
        print(f"Move disk {disk} from A to C")
```

If this is not met, the function repeatedly calls itself, representing the movement of disks between the pegs until the base case is reached. The else statement is repeatedly executed even after moving the largest disk from the source peg to the destination peg, as it continues to move the remaining disks from the auxiliary peg to the destination peg.



```
else:
    # move n-1 disks from A to B
    hanoiSol(n - 1, src, dest, aux)

    # move the largest disk from A to C
    disk = src.pop()
    dest.append(disk)
    print(f"Move disk {disk} from A to C")
```

This activity helped me understand the benefits of using recursion in solving problems, just like the Tower of Hanoi. Recursion makes the solution more concise by reducing the usage of multiple loops and conditions. It also makes the code easier to read and understand.