Programming with Python April 01 2025

Note: Brief Summary of contents discussed.

Recursion: A programming concept, where a function calls itself to solve a smaller version of a problem.
Typically it consist of:

1. Base case: - The condition that stops the recursion.
2. Recursive case: - The function calls itself with a modified argument.

Each recursive call creates a new stack frame. If recursion is too deep, Python raises a recursion error.

Note: Recursive functions can be inefficient for large inputs (e.g. Fibonacci problem has overlapping subproblems resulting in repetitive computation of the same subproblems).

Terms: Tail Recursion, Tree Recursion, Backtracking with Recursion
Example: Factorial using recursion (tail recursion)

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n-1)
print(factorial(5))
```

Example: power(x, n)

```
def power(x, n):
    if n == 0:
        return 1
    else:
        return x * power(x, n-1)
```

Example: Fibonacci using recursion

```
def fibonacci(n):
    if n ==0 or n == 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)
```

Example: Sum of digits of a number

Example: Reverse a string

Example: Generating all binary strings of length n (Tree recursion: Each call branches into two calls, creating a binary tree like structure)

```
    def bin_string(n, s=''):
        if n==0:
            print(s)
            return
        bin_string(n-1, s+'0')
        bin_string(n-1, s+'1')
```

Example: GCD(HCF) of a and b

```
    def gcd(a, b):
        if b%a == 0:
            return a
        return gcd(b%a, a)
```

Example: Tower of Hanoi

```
    def move_disks(src, proxy, dst, n):
        if n == 0:
            return
        else:
            move_disks(src, dst, proxy, n-1)
            print(f'move {n} from {src} to {dst}')
            move_disks(proxy, src, dst, n-1)

    move_disks('X', 'Y', 'Z', 3)
```