Recursion

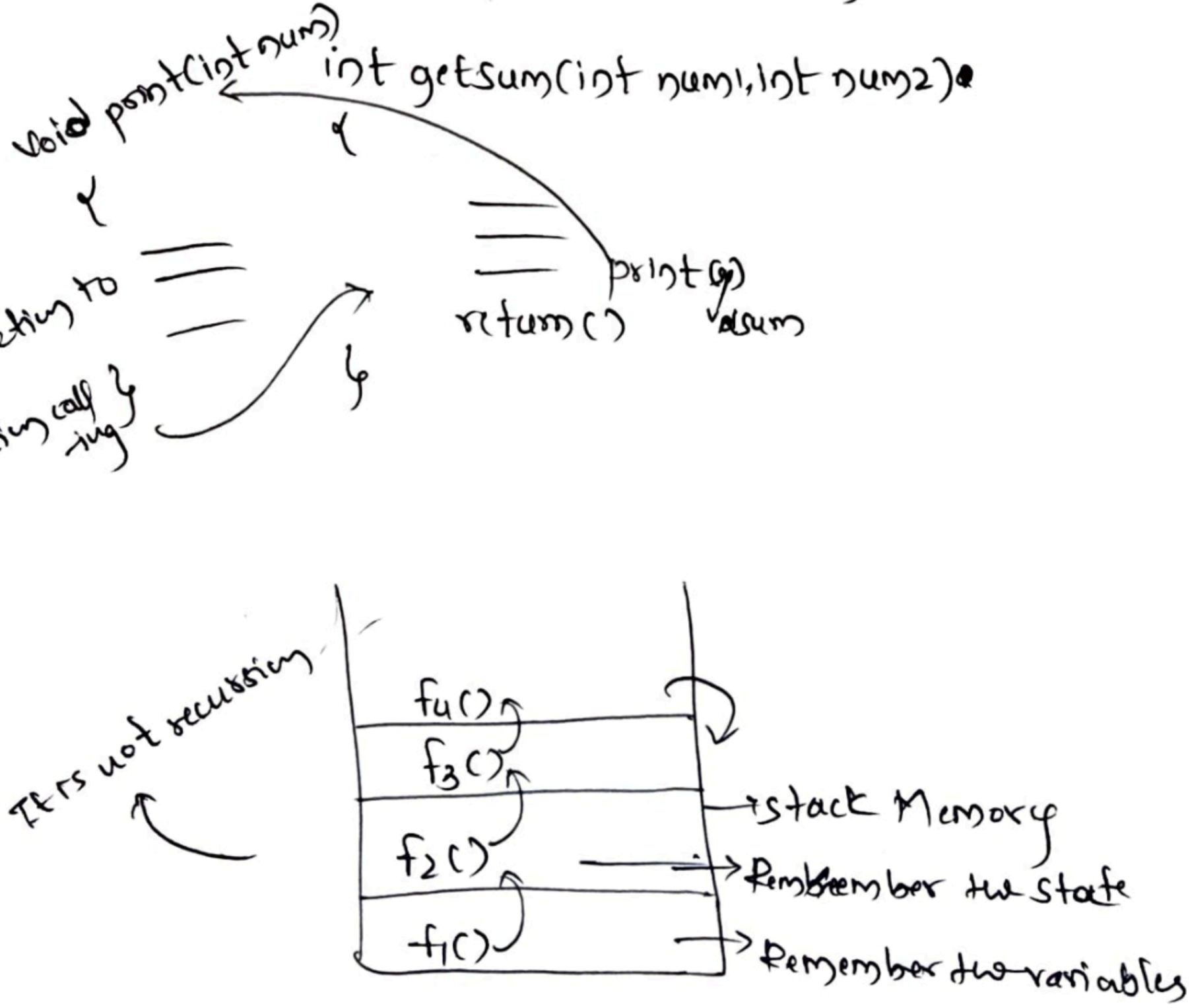
1. Functions
2. Stack
3. Why and where recursion is used
   a) Folder structure
   b) Family tree
4. Important techniques to write recursion
5. Common pitfalls of recursion
6. Seeing recursion live demo
7. Recursion Infinite loop / stack overflow demo / sun
8. Code demo with different example

---

1 Functions (Revision)

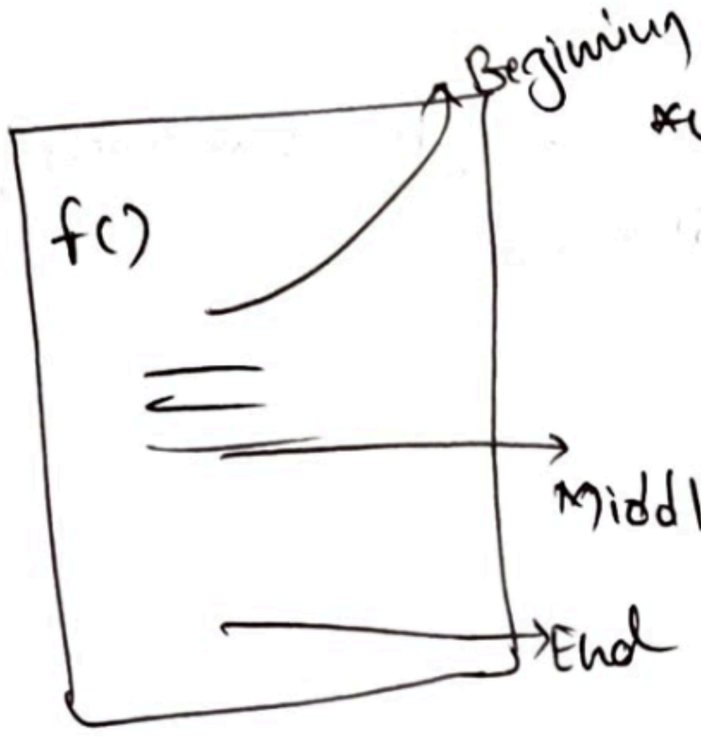
Syntax:

<return type> functionName(arguments)



recursion → same function calling itself.

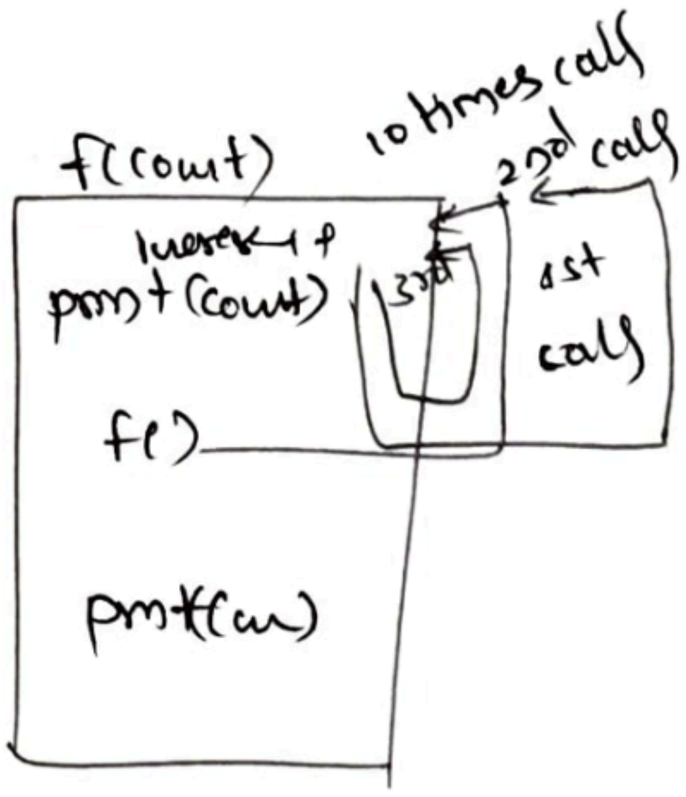
Remembering recent state

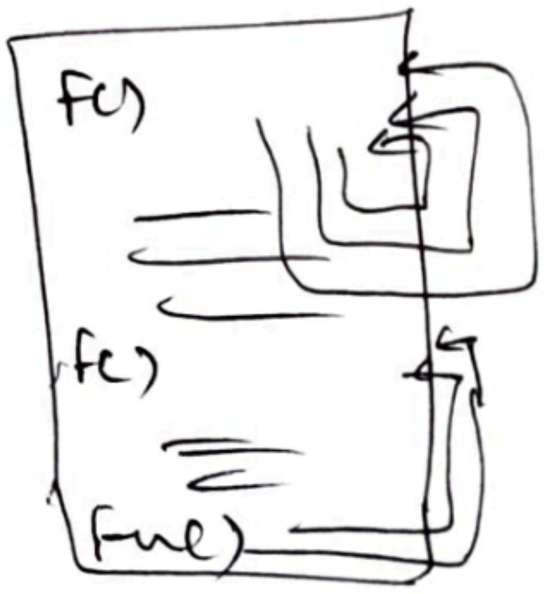


*we can call function from middle, end etc where you calling from is most imp.

② Stack

*Example we are calling from middle the upside of function is filling the stack and the function above is seen while filling and below part removing the variables from stack



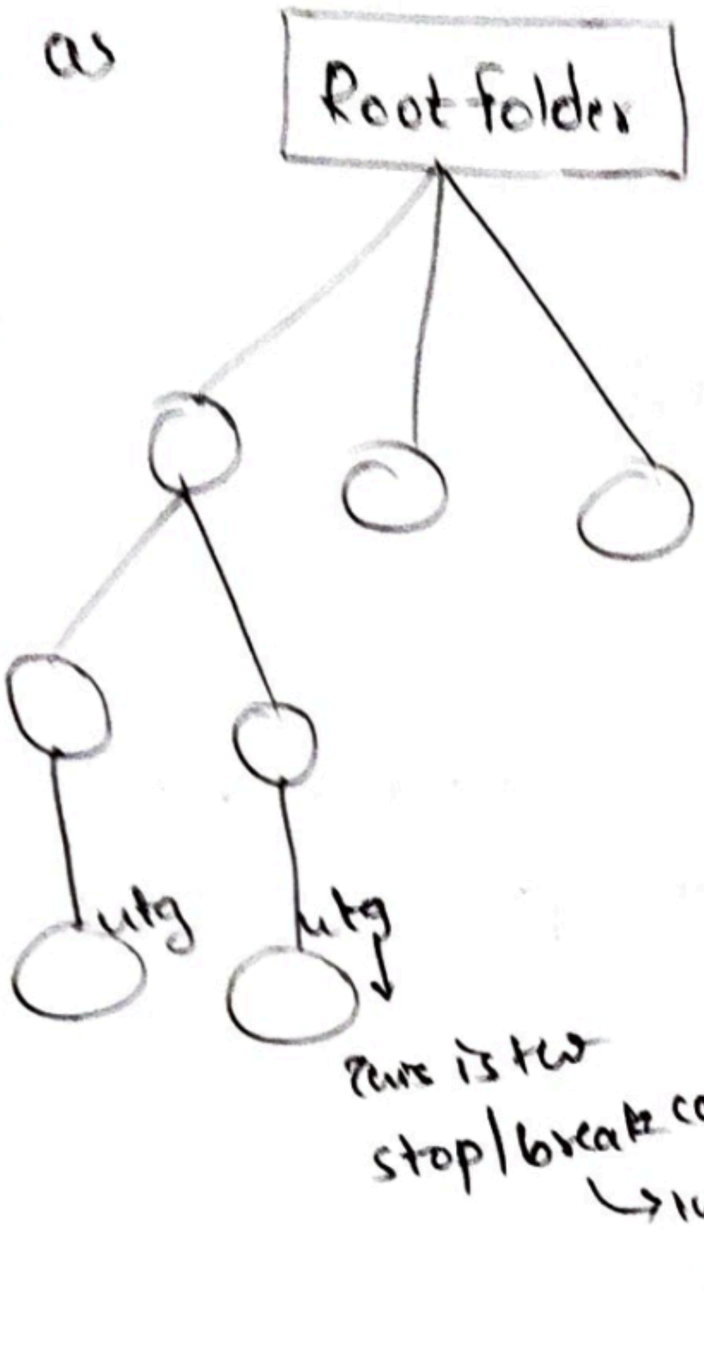


double recursive calling

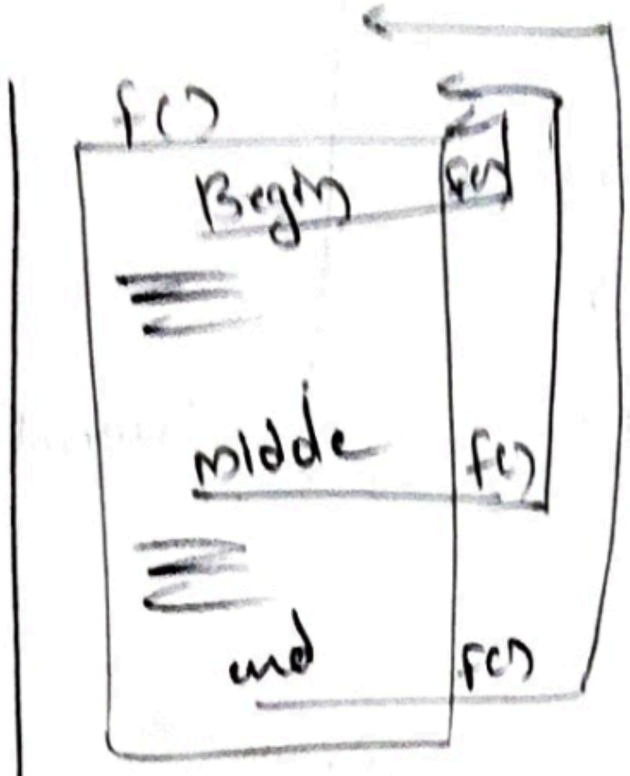
3. Why and where recursion is used.

a) Folder structure
b) family tree.

a)



Base is the stop/break condition
↳ infinite loop



where ur calling is important

* how many times recursion you want call is imp

Recursion

1. Functions and the Stack

Recursion is a programming concept where a function calls itself to solve a problem. It relies on the call stack, a data structure that keeps track of the active function calls. Each time a function is called, a new frame is pushed onto the stack. When the function finishes, its frame is popped off. This process is how the computer manages recursive calls.

2. Why and Where Recursion is Used

Recursion is ideal for problems that can be broken down into smaller, similar subproblems.

Folder Structure: Traversing a file system is a classic example. To list all files in a directory, you can write a recursive function. It checks the current folder, and for any subfolders it finds, it calls itself on that subfolder.

Family Tree: Finding all ancestors of a person is another perfect use case. A recursive function can find a person's parents, and then call itself on each parent to find their parents, and so on, until it reaches the end of the lineage.

3. Important Techniques and Common Pitfalls

Writing a recursive function requires two essential parts: a base case and a recursive step.

Base Case: This is the condition that stops the recursion. Without it, the function will call itself infinitely.

Recursive Step: This is where the function calls itself with a modified input that moves closer to the base case.

A common pitfall is a stack overflow, which happens when a recursive function lacks a base case. It creates an infinite loop of calls, pushing frames onto the stack until memory runs out.

4. Code Demos with Examples

Here are code examples for two common recursive problems: **calculating a factorial** and the **Fibonacci sequence.**

Factorial Calculation :

A factorial (n) is the product of all positive integers up to n. The recursive formula is $n=n\times(n-1)$. The base case is $0=1$.

```
def factorial(n):
    # Base case: if n is 0, the factorial is 1
    if n == 0:
        return 1
    # Recursive step: n * factorial of (n-1)
    else:
```

```
    return n * factorial(n - 1)


print(factorial(5))  # Output: 120 (5 * 4 * 3 * 2 * 1)
```

Fibonacci Sequence :

The Fibonacci sequence is where each number is the sum of the two preceding ones. The base cases are F(0) = 0 and F(1) = 1. The recursive step is F(n)=F(n−1)+F(n−2).

```
def fibonacci(n):
    # Base cases: for n=0 and n=1
    if n <= 1:
        return n
    # Recursive step: sum of the two preceding numbers
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)


print(fibonacci(7))  # Output: 13 (0, 1, 1, 2, 3, 5, 8, 13)
```

NonRecursive_lyst1726605695363.py

C: > recursive funtions > NonRecursive_lyst1726605695363.py > ...

```python

def incrementPrint1(value):
    value += 1
    print(f"incrementPrint1: {value}")
    incrementPrint2(value)

def incrementPrint2(value):
    value += 1
    print(f"incrementPrint2: {value}")
    incrementPrint3(value)

def incrementPrint3(value):
    value += 1
    print(f"incrementPrint3: {value}")
    incrementPrint4(value)

def incrementPrint4(value):
    value += 1
    print(f"incrementPrint4: {value}")
    incrementPrint5(value)

def incrementPrint5(value):
    value += 1
    print(f"incrementPrint5: {value}")
    incrementPrint6(value)

def incrementPrint6(value):
    value += 1
    print(f"incrementPrint6: {value}")
```

PROBLEMS DEBUG CONSOLE TERMINAL PORTS

TERMINAL

```
incrementPrint6: 6
incrementPrint7: 7
incrementPrint8: 8
incrementPrint9: 9
incrementPrint10: 10
PS C:\recursive funtions>
```

File Edit Selection View Go Run Terminal Help Search

NonRecursive_lyst1726605695363.py FolderExplore_lyst1726605695362.py FamilyTree_lyst1726605695362.py

C: > recursive funtions > FamilyTree_lyst1726605695362.py > ...

```python

class Node:
    def __init__(self, name):
        self.name = name
        self.father = None

def create_family_tree_hardcoded():
    # Hardcoded family tree
    names = ["Pralhad", "Hiranyakashapa", "Kashayapa Brahma", "Chaturmukha Brahma", "I don't know"]

    # Create nodes from the names list
    nodes = [Node(name) for name in names if name.lower() != "i don't know"]

    # Link nodes to form the tree
    for i in range(len(nodes) - 1):
        nodes[i].father = nodes[i + 1]

    # Return the root of the tree (the last generation individual)
    return nodes[0]

def print_family_tree(node):
    if node is not None:
        print(f"{node.name}")
        print_family_tree(node.father)

# Example usage
if __name__ == "__main__":
    root = create_family_tree_hardcoded()
    print("Family Tree:")
```

PROBLEMS DEBUG CONSOLE TERMINAL PORTS

TERMINAL

```
Family Tree:
Pralhad
Hiranyakashapa
Kashayapa Brahma
Chaturmukha Brahma
PS C:\recursive funtions>
```

File Edit Selection View Go Run Terminal Help

Search

NonRecursive_lyst1726605695363.py

FolderExplore_lyst1726605695362.py

C: > recursive funtions > FolderExplore_lyst1726605695362.py > ...

```python

import os

def explore_directories(root_directory):
    try:
        # List all the entries in the directory
        entries = os.listdir(root_directory)
        for entry in entries:
            path = os.path.join(root_directory, entry)
            if os.path.isdir(path):
                print(path)
                # Recursively explore the sub-directory
                explore_directories(path)
    except PermissionError:
        # Handle the case where the program doesn't have permission to access a directory
        print(f"Permission denied: {root_directory}")
    except FileNotFoundError:
        # Handle the case where a directory was not found
        print(f"Directory not found: {root_directory}")

# Example usage
root_directory = "C://RecurssionDemo"  # Replace with the root directory path you want to explore
explore_directories(root_directory)

```