Module-5

Recursion, Stack, Queue

Recursion

When a function calls itself, it’s called recursion.

How to write a recursive program?

1. Define base condition (where to stop)
2. Define when to continue while modifying the parameters while calling the function.

Recursive program

```
def factorial(number)
{       if number == 1:  // Base condition (where to stop)
                return 1
        else:
                number*factorial(number-1) // when to continue and modify parameters
}
```

Different Types of Recursion

1. **Tail Recursion** - The recursive call is the last operation - Nothing executes after the recursive call returns and No pending operations - The function doesn't need to remember anything for after the recursive call
2. **Non Tail Recursion** - Non-tail recursion is a type of recursion where the recursive call is not the last operation in the function. This means the function performs some additional operations after the recursive call returns.
3. **Indirect Recursion**
4. **Nested Recursion**

Tail Recursion - Print the array elements

```
def print_array(arr,i,j):
{    if i == j : // Base condition (where to stop)
        print(arr[i])
        return
    else:
        print(arr[i])
        print_array(arr,i+1,j) // only 1 function call that too at the end of the program
}
```

Non Tail Recursion

```
def factorial(number)
{      if number == 1:  // Base condition (where to stop)
            return 1
       else:
            number*factorial(number-1) // when to continue and modify parameters
}
```

Multiplication happens after recursion

Indirect Recursion

Indirect recursion occurs when a function calls another function, which in turn calls the first function (or a chain of functions that eventually leads back to the original). Unlike direct recursion where a function calls itself, indirect recursion involves multiple functions mutually calling each other in a cycle.

python

```python
def is_even(n):
    if n == 0:
        return True
    else:
        return is_odd(n - 1)

def is_odd(n):
    if n == 0:
        return False
    else:
        return is_even(n - 1)

print(is_even(4))  # True
print(is_odd(5))   # True
```

Nested Recursion

Nested recursion is a special form of **recursion where a function calls itself with its own recursive call as an argument**. This creates a "nested" effect, where the recursion happens at multiple levels simultaneously.

```
def ackermann(m, n):
    if m == 0:
        return n + 1
    elif n == 0:
        return ackermann(m - 1, 1)
    else:
        return ackermann(m - 1, ackermann(m, n - 1))  # Nested recursion!
```

Stack Implementation using array

ADT of Stack : tells only about the operations possible on stack, not the internal details

1. push()
2. pop()

Implementation of stack - means write code of push and pop using array.

See notebook for the code.

Queue Implementation using array

ADT of Queue : tells only about the operations possible on stack, not the internal details

1. enqueue()
2. dequeue()

Implementation of queue - means write code of enqueue() and dequeue() using array.

See notebook for the code.