**ASSIGNMENT 1**

**WRITE PSEUDOCODE**

Step1: Start

Step2: initialize variables one for input and other for result

Step3: ask the user for input and store it in input variable

Step4: check for the number to be positive

If the number is less than 0 return message “number should be positive”

Else continue to the next step

Step4: check if the number to be even

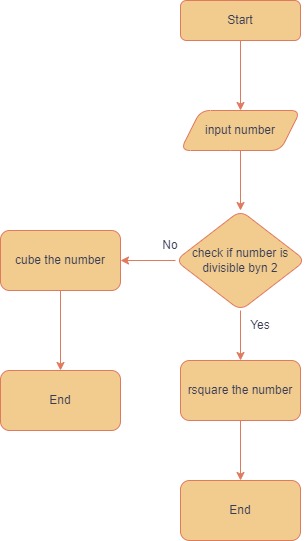
Calculate the square of the input number and store it in the ‘result’ variable

Else if the number is odd

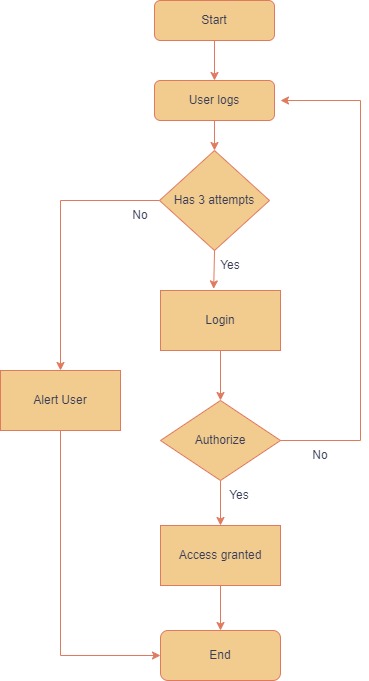
Calculate the cube of the input number and store it in the ‘result’ variable

Step5: display the result to the user

Step6: End



**Assignment 2 Flow chart**



**Pseudo code for Login System**

procedure userLogin()

// Define variables

let Attempts = 3

let attemptCount = 0

let isLoggedIn = false

start the while loop to count for number of attempts

while attemptCount < Attempts AND isLoggedIn equals false do:

// Prompt user for username and password

input username

input password

if username equals correctUsername AND password equals correctPassword then:

isLoggedIn = true

output "Login successful. Welcome, ", username

else:

// Increment attempt count

attemptCount = attemptCount + 1

output "Login failed. You have ", Attempts - attemptCount, " attempts remaining."

if attemptCount = Attempts then:

output "Maximum login attempts reached. Your account is locked."

Exit loop

end procedure

**Question 3**

**Function design and modularization**

We will go with factorial function lets write the pseudo code for it

function factorial(n)

IF n == 0

return 1

else

result = 1

for i = 1 to n

result = result \* i

end for

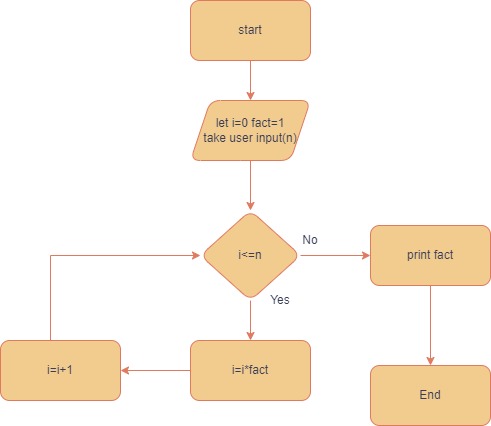
retrurn result

end if

end function

The factorial function takes a single parameter 'n' representing the number whose factorial needs to be calculated. If 'n' is equal to 0, the function returns 1 (as 0! equals 1). Otherwise, it iterates from 1 to 'n', multiplying each number in the range to calculate the factorial, and then returns the result.

**FLOW CHART**



**Now for Fibonacci series**

**Here n is the number of items in the series we can also do it by specifying upto which number we want the series**

function Fibonacci(n):

if n <= 1:

return n

else:

prev = 0

current = 1

for i from 2 to n:

next = prev + current

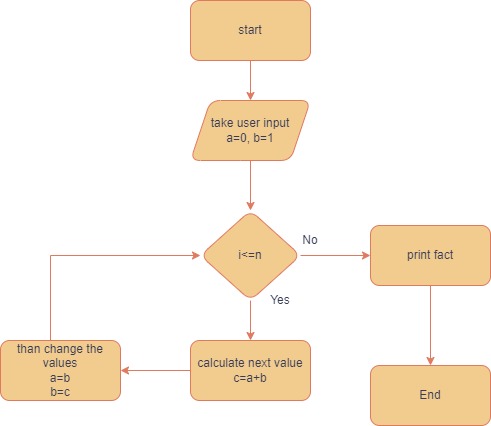
prev = current

current = next

return current

The fibonacci function takes a single parameter 'n' representing the position of the Fibonacci number to be calculated. If 'n' is less than or equal to 1, the function returns 'n' itself. Otherwise, it initializes two variables prev and current with the first two Fibonacci numbers (0 and 1) and iterates from 2 to 'n', calculating each subsequent Fibonacci number by summing the two preceding on

**FLOW CHART**

es.

**MODULARITY**

Modularity is defined as breaking down the program into smaller modules of functions that perform specific tasks , in return this will make the code more organized and readable

We can say code reuse in the sense as certain type functions are kept in certain files which will make the code more reusable as we can fetch the function directly from that file