



**SNS COLLEGE OF ENGINEERING**  
Kurumbapalayam (Po), Coimbatore – 641 107

**An Autonomous Institution**

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade  
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai



**DEPARTMENT OF COMPUTER SCIENCE AND DESIGN**



# **19IT103 – COMPUTATIONAL THINKING AND PYTHON PROGRAMMING**

❖ A readable, dynamic, pleasant, flexible, fast and powerful language

## Recap:

- An algorithm is **a sequence of non ambiguous instructions** for solving a problem in a finite amount of time.
- An input to an algorithm specifies an instance of the problem the algorithm solves.
- Algorithm can be specified in a natural language or a pseudocode; they can also be implemented as computer programs.

## Recap:

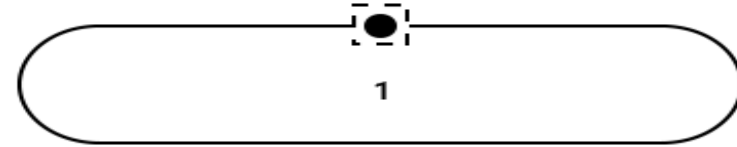
- *Algorithm design techniques* are general approaches to solving problems algorithmically, applicable to a variety of problems from different areas of computing.
- **The same problem can often be solved by several algorithms.**
- Algorithms operate on data. This makes the issue of data structuring critical for efficient algorithmic problem solving.

## 1.7 Simple strategies for developing algorithms:

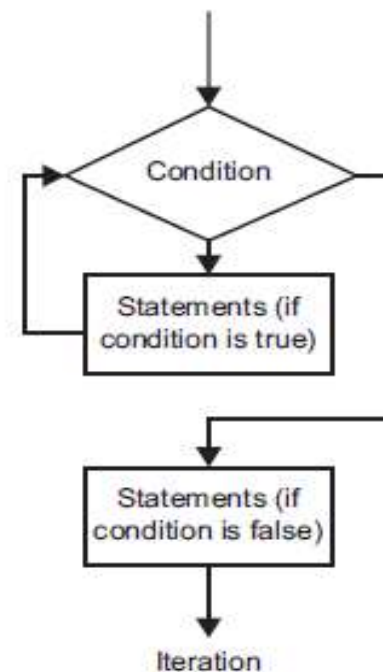
- An algorithm is a defined set of step-by-step procedures that provides the correct answer to a particular problem.
- There are some simple strategies for developing algorithms:
  - **Iteration**
  - **Recursion**
  - Brute force.
  - Backtracking.
  - Greedy Method (Heuristics)
  - Divide and Conquer.
  - Dynamic Programming.
  - Branch and Bound.

## 1.7 Simple strategies for developing algorithms:

### 1.7.1 Iteration:



- A sequence that is executed repeatedly so long as a certain condition holds.
- A sequence of statements is executed **until a specified condition is true** is called *iterations*.
  - for loop
  - while loop



## 1.7 Simple strategies for developing algorithms:

### 1.7.1 Iteration:

#### for loop:

- The **for-loop** sets up a **control variable** that manages execution of the loop.
- Execution iterates over the items in a sequence (the value of each item is assigned to the control variable at the beginning of each pass through the loop).
- That sequence could, for example, be a list.
- In the following code sample, the variable *word* is used as a control variable.
- At the beginning of each iteration of the loop, it is assigned the next value from the list *words* from beginning to end.

## 1.7 Simple strategies for developing algorithms:

### 1.7.1 Iteration:

for loop:


- Syntax of for loop:

FOR( start-value to end-value) DO

Statement

...

ENDFOR

 Wikitechy.com <pre>for ( a = 1;    a &lt; 5;    a ++ ) {     printf( "%d", a ); }</pre>	<table><thead><tr><th>a</th><th>Output</th></tr></thead><tbody><tr><td>1</td><td></td></tr></tbody></table>	a	Output	1	
a	Output				
1					

## 1.7 Simple strategies for developing algorithms:

### 1.7.1 Iteration:

- for loop: example 1:
- # This prints out the length of each word in a list of words

```
words = ['my', 'big', 'meal', 'comes', 'mostly', 'bearing', 'doubtful',  
'garnishes']
```

```
for word in words:
```

```
    # The following line prints the length of the word
```

```
        print(len(word))
```

```
# Prints: 2 3 4 5 6 7 8 9
```



## 1.7 Simple strategies for developing algorithms:

### 1.7.1 Iteration:

- for loop: example 2:
- if you know exactly how many iterations to execute, a range:

```
for number in range(1, 13):
```

```
    print(number * 42)
```

```
    # Prints out the 42 times table
```

## 1.7 Simple strategies for developing algorithms:

### 1.7.1 Iteration:

- for loop: example 3: Print  $n$  natural numbers

BEGIN

GET n

INITIALIZE i=1

FOR (i<=n) DO

PRINT i

i=i+1

ENDFOR

END

## 1.7 Simple strategies for developing algorithms:

### 1.7.1 Iteration:

#### While loop:

- The while loop **executes a block of instructions repeatedly** for as long as some condition evaluates to true.
- The **value of the condition is only checked at the beginning of each iteration.**
- As soon as the condition evaluates to false, the loop ends and execution jumps immediately to the next line following the end of the while block.

## 1.7 Simple strategies for developing algorithms:

### 1.7.1 Iteration:

#### While loop:

- Syntax of while loop:

WHILE (condition) DO

Statement

...

ENDWHILE

code	output
1 a = 1	
2 while a < 10:	
3 print (a)	
4 a += 2	
variables	

## 1.7 Simple strategies for developing algorithms:

### 1.7.1 Iteration:

#### While loop: example 1:

- #This program invites the user to guess a number (set in the # age variable). As long as they haven't guessed correctly, the program keeps asking.

```
age = 25
```

```
guess = 0
```

```
while age != guess:
```

```
    # Whereas a == b tests whether a and b are equal, a != b tests whether a and b are not equal
```

```
    # The int() function turns the user's input (which is text) into an integer.
```

```
        guess = int(input('Guess how old I am> '))
```

```
print('You got it right!')
```

## 1.7 Simple strategies for developing algorithms:

### 1.7.1 Iteration:

While loop: example 2: Print  $n$  natural numbers :

BEGIN

GET  $n$

INITIALIZE  $i=1$

WHILE( $i \leq n$ ) DO

PRINT  $i$

$i=i+1$

ENDWHILE

END

## 1.7 Simple strategies for developing algorithms:

### 1.7.1 Iteration:

While loop: example 3: To find power of a number :

TASK: To Find Power of a number

READ number

READ Power

Initialize result with number and pow with Power

WHILE pow < Power:

    result = result \* number

    Increase pow by 1

End Loop

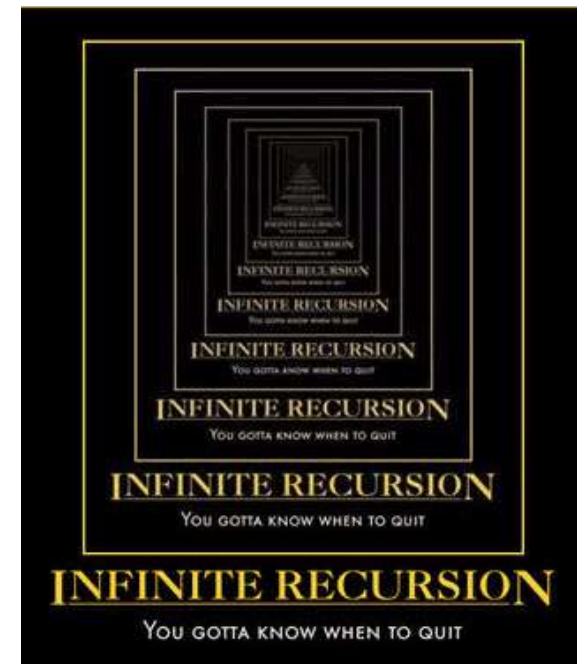
PRINT result

End

## 1.7 Simple strategies for developing algorithms:

### 1.7.2 Recursion:

- **A function that calls itself** is known as *recursion*.
- Recursion is a process by which a function calls itself repeatedly until some specified condition has been satisfied.
- A physical world example would be to **place two parallel mirrors facing each other**. Any object in between them would be reflected recursively.





## 1.7 Simple strategies for developing algorithms:

### 1.7.2 Recursion:

- Python Recursive Function

```
def recurse():  
    ...  
    recurse()  
    ...  
recurse()
```

recursive call

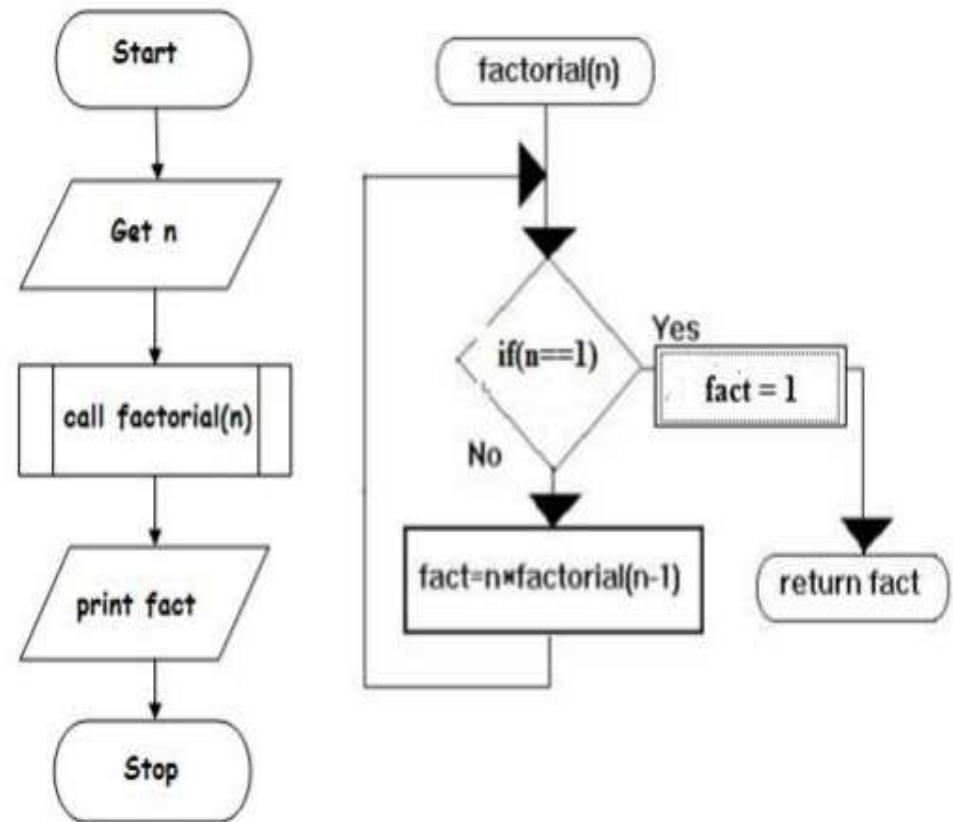
## 1.7 Simple strategies for developing algorithms:

### 1.7.2 Recursion:

- Algorithm for factorial of **n** numbers using recursion:

#### Main function:

- Step1: Start
- Step2: Get n
- Step3: call factorial(n)
- Step4: print fact
- Step5: Stop



#### Sub function factorial(n):

- Step1: if(n==1) then fact=1 return **fact**
- Step2: else fact=n\*factorial(n-1) and return **fact**

## 1.7 Simple strategies for developing algorithms:

### 1.7.2 Recursion:

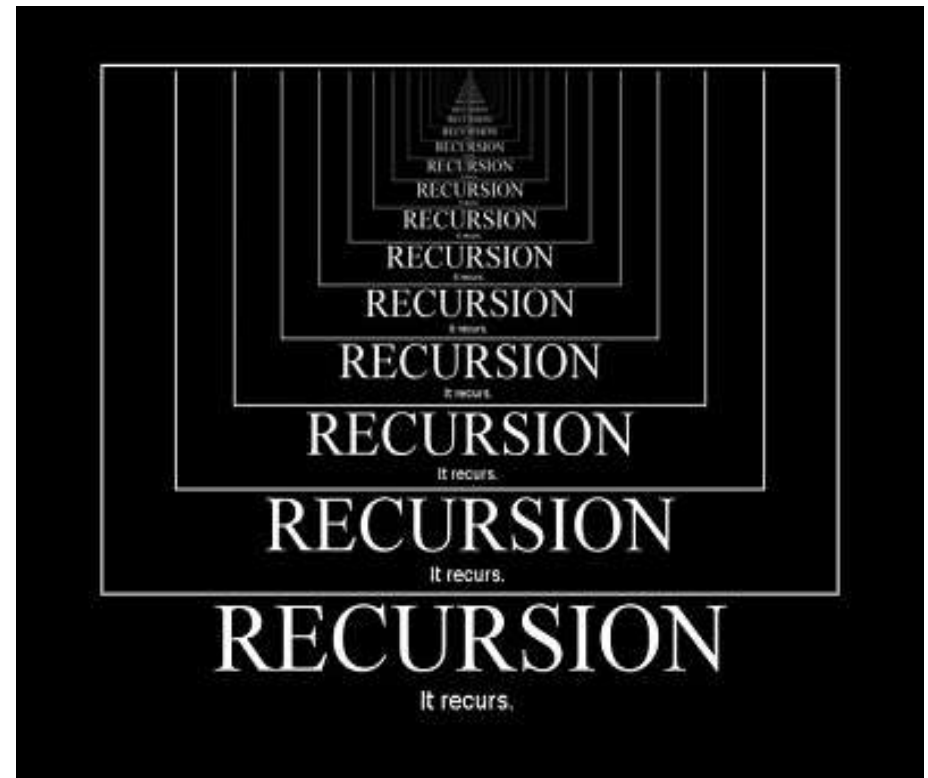
- Pseudo code for factorial using recursion:

#### Main function:

```
BEGIN  
GET n  
CALL factorial(n)  
PRINT fact  
END
```

#### Sub function factorial(n):

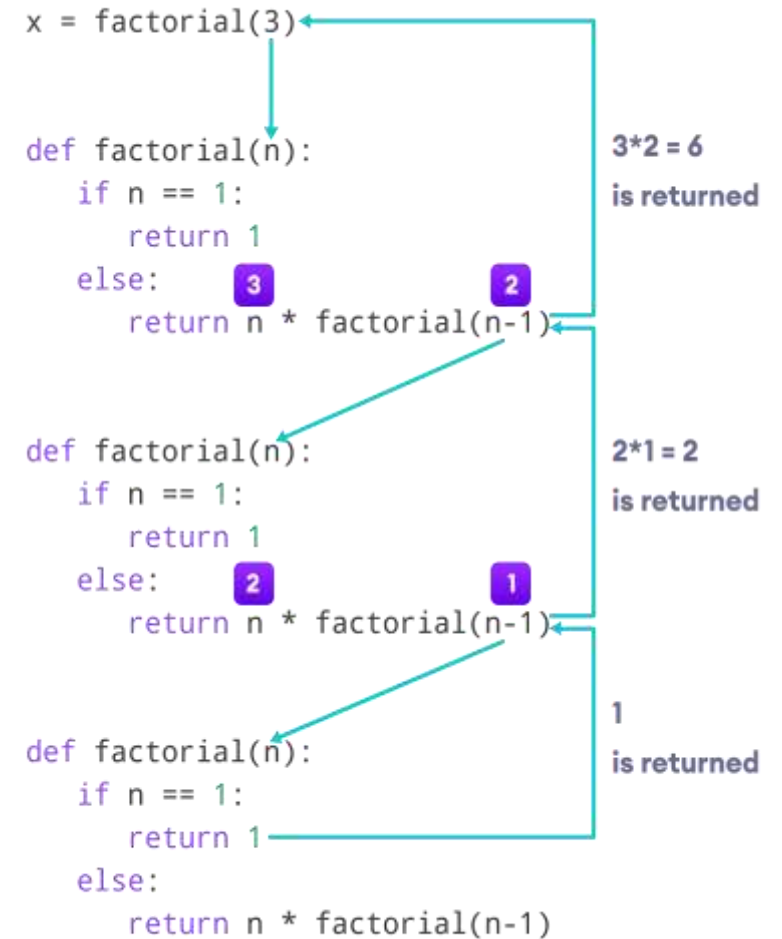
```
IF(n==1) THEN  
    fact=1  
    RETURN fact  
ELSE  
    RETURN fact=n*factorial(n-1)
```



# 1.7 Simple strategies for developing algorithms:

## 1.7.2 Recursion:

factorial(3)            # 1st call with 3  
3 \* factorial(2)        # 2nd call with 2  
3 \* 2 \* factorial(1)    # 3rd call with 1  
3 \* 2 \* 1               # return from 3rd call as number=1  
3 \* 2                    # return from 2nd call  
6                         # return from 1st call



## 1.7 Simple strategies for developing algorithms:

### 1.7.2 Recursion:

```
factorial( n ):
```

```
if n == 1:  
    return 1
```

```
else:
```

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

```
        if n == 1:  
            return 1  
        else:
```

---

*factorial(n) =*

## 1.7 Simple strategies for developing algorithms:



### 1.7.2 Recursion:

#### Advantages of Recursion:

- Recursive functions **make the code look clean** and **elegant**.
- **A complex task can be broken down into simpler sub-problems** using recursion.
- **Sequence generation is easier** with recursion than using some nested iteration.

## 1.7 Simple strategies for developing algorithms:

### 1.7.2 Recursion:

#### Disadvantages of Recursion:

- Sometimes the **logic behind recursion is hard to follow** through.
- Recursive calls are expensive (inefficient) as they **take up a lot of memory and time.**
- Recursive functions are **hard to debug.**



## Summary:

- Simple strategies for developing algorithms:
  - Iteration
  - Recursion
- Iteration: **A sequence that is executed repeatedly so long as a certain condition holds.** A sequence of statements is executed until a specified condition is true is called iterations.
  - for loop
  - While loop
- Recursion: **A function that calls itself is known as recursion.**
- Recursion is a process by which a function calls itself repeatedly until some specified condition has been satisfied.



A yellow speech bubble with a pointed tail pointing towards the bottom right, set against a solid blue background. The words "THANK YOU" are cut out of the bubble in a bold, sans-serif font, revealing the blue background behind them.

**THANK YOU**