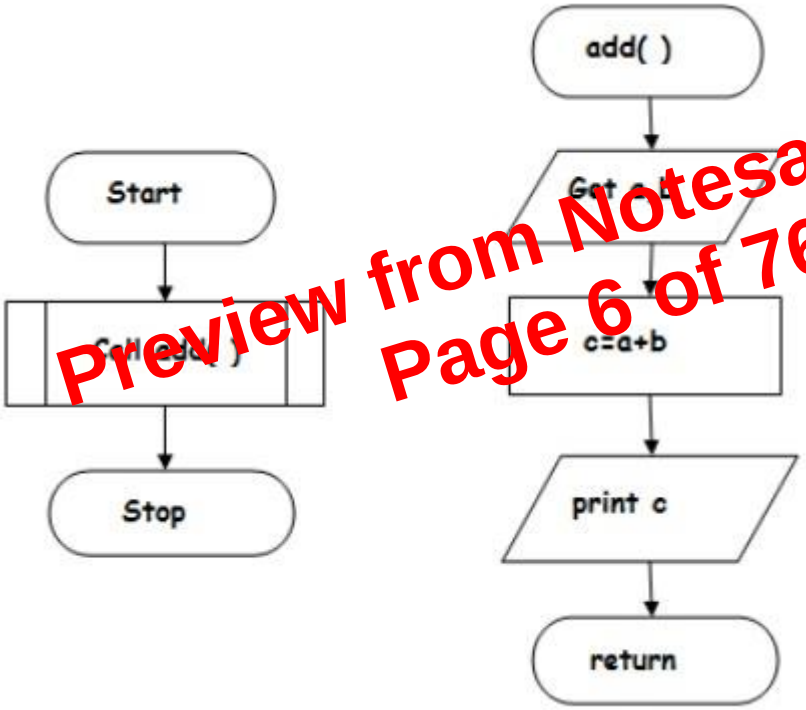
Example:

Algorithm for addition of two numbers using function
Main function()
Step 1: Start
Step 2: Call the function add()
Step 3: Stop

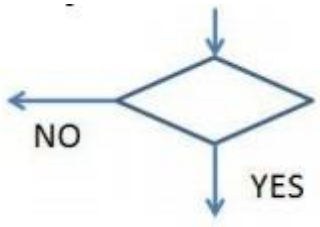
sub function add()
Step 1: Function start
Step 2: Get a, b Values
Step 3: add c=a+b
Step 4: Print c
Step 5: Return



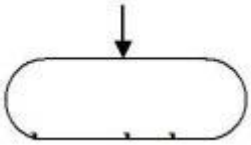
NOTATIONS

FLOW CHART

Flow chart is defined as graphical representation of the logic for problem solving.
The purpose of flowchart is making the logic of the program clear in a visual representation.



5. Only one flow line is used with a terminal symbol.



6. Within standard symbols, write briefly and precisely.
7. Intersection of flow lines should be avoided.

Advantages of flowchart:

1. **Communication: -** Flowcharts are better way of communicating the logic of a system to all concerned.
2. **Effective analysis: -** With the help of flowchart, problem can be analyzed in more effective way.
3. **Proper documentation: -** Program flowcharts serve as a good program documentation, which is needed for various purposes.
4. **Efficient Coding: -** The flowcharts act as a guide or blueprint during the system analysis and program development phase.
5. **Proper Debugging: -** The flowchart helps in debugging process.
6. **Efficient Program Maintenance: -** The maintenance of operating program becomes easy with the help of flowchart. It helps the programmer to put efforts more efficiently on that part.

Disadvantages of flow chart:

1. **Complex logic: -** Sometimes, the program logic is quite complicated. In that case, flowchart becomes complex and clumsy.
2. **Alterations and Modifications: -** If alterations are required the flowchart may require re-drawing completely.
3. **Reproduction: -** As the flowchart symbols cannot be typed, reproduction of flowchart becomes a problem.
4. **Cost: F**or large application the time and cost of flowchart drawing becomes costly.

Functional programming language:

Functional programming language defines every computation as a mathematical evaluation. They focus on the programming languages are bound to mathematical calculations

Examples:

Clean

Haskell

Compiled Programming language:

A compiled programming is a programming language whose implementation are typically compilers and not interpreters.

It will produce a machine code from source code.

Examples:

C

C++

C#

JAVA

Procedural programming language:

Procedural (imperative) programming implies specifying the steps that the programs should take to reach to an intended state.

A procedure is a group of statements that can be referred through a procedure call. Procedures help in the reuse of code. Procedural programming makes the programs structured and easily traceable for program flow.

Examples:

Hyper talk

MATLAB

Scripting language:

Scripting language are programming languages that control an application. Scripts can execute independent of any other application. They are mostly embedded in the application that they control and are used to automate frequently executed tasks like communicating with external program.

Examples:

Proving an Algorithm's Correctness

v Once an algorithm has been specified, you have to prove its ***correctness***. That is, you have to prove that the algorithm yields a required result for every legitimate input in a finite amount of time.

v A common technique for proving correctness is to use mathematical induction because an algorithm's iterations provide a natural sequence of steps needed for such proofs.

v It might be worth mentioning that although tracing the algorithm's performance for a few specific inputs can be a very worthwhile activity, it cannot prove the algorithm's correctness conclusively. But in order to show that an algorithm is incorrect, you need just one instance of its input for which the algorithm fails.

Analysing an Algorithm

1. Efficiency.

Time efficiency, indicating how fast the algorithm runs,

Space efficiency, indicating how much extra memory it uses.

2. simplicity.

v An algorithm should be precisely defined and investigated with mathematical expressions.

v Simpler algorithms are easier to understand and easier to program.

v Simple algorithms usually contain fewer bugs.

Coding an Algorithm

v Most algorithms are destined to be ultimately implemented as computer programs. Programming an algorithm presents both a peril and an opportunity.

v A working program provides an additional opportunity in allowing an empirical analysis of the underlying algorithm. Such an analysis is based on timing the program on several inputs and then analysing the results obtained.

SIMPLE STRATEGIES FOR DEVELOPING ALGORITHMS:

1. iterations
2. Recursions

-

1. Iterations:

A sequence of statements is executed until a specified condition is true is called iterations.

1. for loop
2. While loop

Syntax for For:

FOR(*start-value to end-value*) DO
Statement
...
ENDFOR

Example: Print n natural numbers

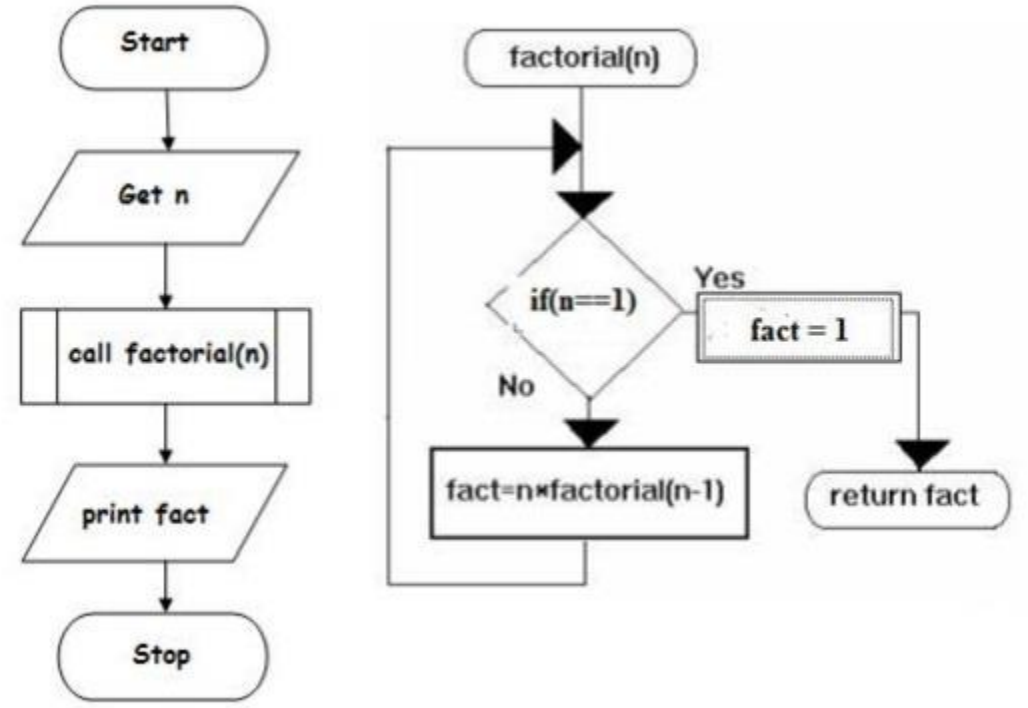
```
BEGIN
GET n
INITIALIZE i=1
FOR (i<=n) DO
PRINT i
i=i+1
ENDFOR
END
```


Syntax for While:

WHILE (condition) DO
Statement
...
ENDWHILE

Example: Print n natural numbers

```
BEGIN
GET n
INITIALIZE i=1
WHILE(i<=n) DO
PRINT i
i=i+1
ENDWHILE
END
```



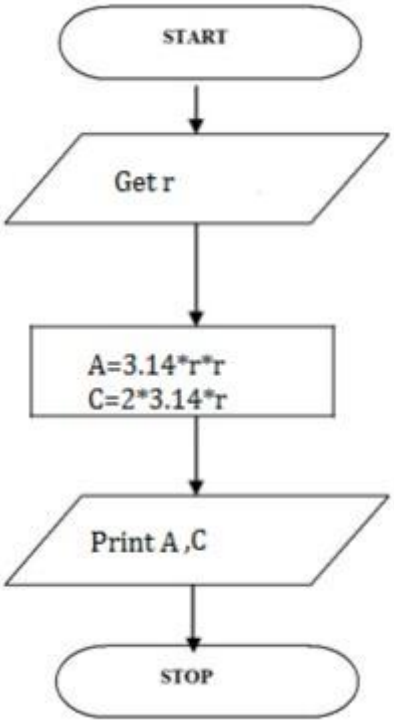
Pseudo code for factorial using recursion:

Main function:
BEGIN
GET n
CALL factorial(n)
PRINT fact
BIN

Sub function factorial(n):
IF(n==1) THEN
    fact=1
    RETURN fact
ELSE
    RETURN fact=n*factorial(n-1)

More examples:

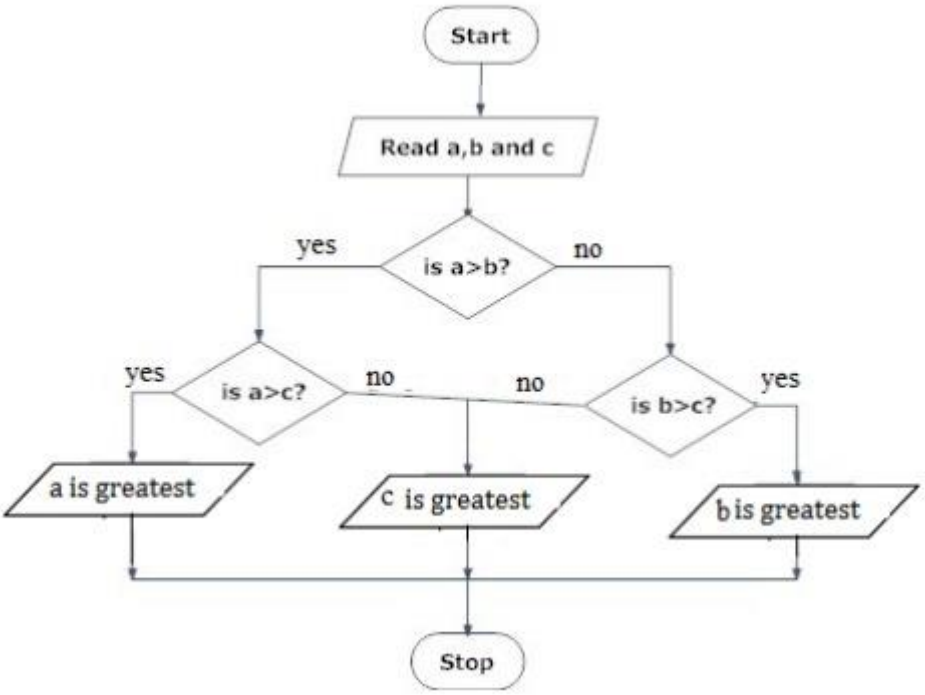
Write an algorithm to find area of a rectangle



BEGIN
READ r
CALCULATE A and C
A=3.14*r*r
C=2*3.14*r
DISPLAY A
END

Write an algorithm for Calculating simple interest

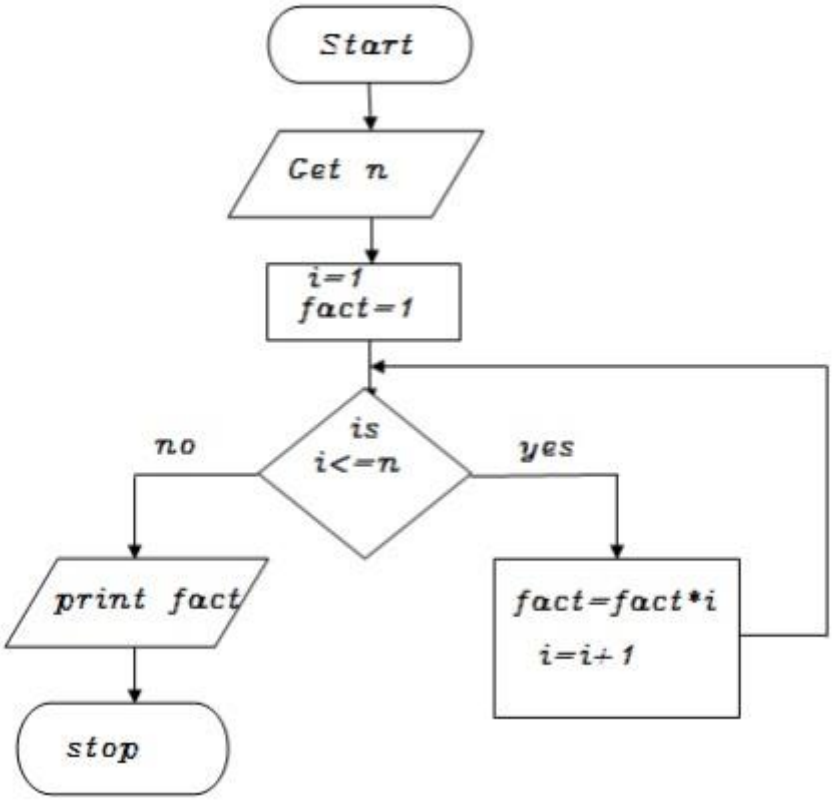
Step 1: Start
Step 2: get P, n, r value
Step3:Calculate
SI=(p*n*r)/100
Step 4: Display S
Step 5: Stop



BEGIN
READ a, b, c
IF (a>b) THEN
IF(a>c) THEN
DISPLAY a is greater
ELSE
DISPLAY c is greater
END IF
ELSE
IF(b>c) THEN
DISPLAY b is greater
ELSE
DISPLAY c is greater
END IF
END IF
END

Write an algorithm to check whether given number is +ve, -ve or zero.

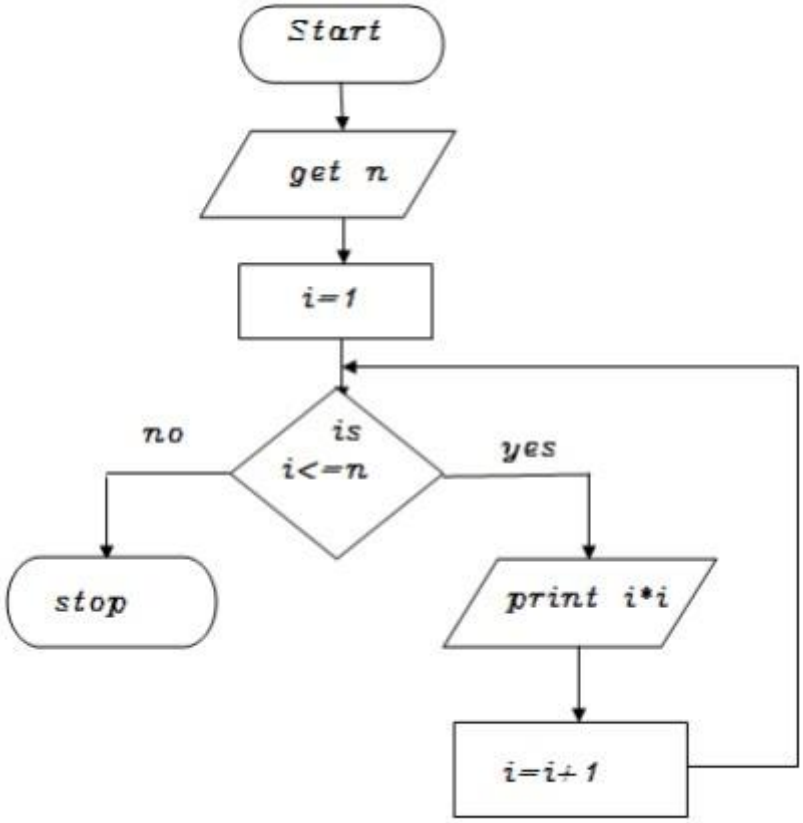
Step 1: Start
Step 2: Get n value.
Step 3: if (n ==0) print "Given number is Zero" Else goto step4
Step 4: if (n > 0) then Print "Given number is +ve"



BEGIN
GET n
INITIALIZE i=1,fact=1
WHILE(i<=n) DO
fact=fact*i
i=i+1
ENDWHILE
PRINT fact
END

Write an algorithm to find area of a rectangle

Step 1: Start
Step 2: get l,b values
Step 3: Calculate A=l*b
Step 4: Display A
Step 5: Stop



BEGIN
GET n
INITIALIZE i=1
WHILE(i<=n) DO
PRINT i*i
i=i+2
ENDWHILE
END

Write an algorithm to print to print cubes of a number

Step 1: start
step 2: get n value
step 3: set initial value i=1
step 4: check i value if(i<=n) goto step 5 else goto step8
step 5: print i*i *i value
step 6: increment i value by 1
step 7: goto step 4
step 8: stop

area of rectangle using function

```
def area():
l=eval(input(“enter the length of rectangle”))
b=eval(input(“enter the breath of rectangle”))
a=l*b
print(“the area of rectangle is”,a)
area()
```

Output

```
enter the length of
rectangle 20
enter the breath of
rectangle 5
the area of rectangle is
100
```

swap two values of variables

```
def swap():
a=eval(input("enter a value"))
b=eval(input("enter b value"))
c=a
a=b
b=c
print("a=",a,"b=",b)
swap()
```


Output

```
enter a value3
enter b value5
a= 5 b= 3
```

check the no divisible by 5 or not

```
def div():
n=eval(input("enter n value"))
if(n%5==0):
print("the number is divisible by 5")
else:
```

```
print("the number not divisible by 5")
div()
```

Output

```
enter n value10
the number is divisible by
5
```

find reminder and quotient of given no

```
def reminder():
a=eval(input("enter a"))
b=eval(input("enter b"))
R=a%b
print("the reminder is",R)
def quotient():
a=eval(input("enter a"))
b=eval(input("enter b"))
Q=a/b
print("the reminder is",Q)
reminder()
quotient()
```


Output

```
enter a 6
enter b 3
the reminder is 0
enter a 8
enter b 4
the reminder is 2.0
```

convert the temperature

```
def ctof():
c=eval(input("enter temperature in centigrade"))
f=(1.8*c)+32
print("the temperature in Fahrenheit is",f)
def ftoc():
f=eval(input("enter temp in Fahrenheit"))
```