UNIT - 1

Computational Thinking And Problem Solving

Fundamentals of computing - Identification of Computational problems - Algorithms, building blocks of algorithms (statements, state, control flow, functions), notation (Pseudo code, flow chart, programming language), algorithms problem solving, simple strategies for developing algorithms (iteration, recursion). Illustrative problems: Find minimum in a list, insert a card in a list of sorted cards, guess an integer number in a range Towers of Hanoi.

Computer

Computer is an electronic device that performs tasks according to a set of instructions or programs.

To perform the tasks, the computer follows three stages or phases

i) Input

ii) Process

iii) Output.

Input

The data entered in a computer is called as the input.

Process

Manipulation of data is called as process.

Output

The processed data usually called as information is obtained as output.

Computer Hardware

The components or physical elements that we can touch and see are called the hardware components of a computer system.

They include CPU, Main memory, Secondary Memory, Input/output devices

Central Processing Unit (CPU)

The central Processing Unit is called the Brain of a computer system, since it contains all the digital logic circuits necessary to interpret and execute instructions.

Main memory

Main memory is where currently executing programs reside which the CPU can directly and very quickly access.

Main memory is volatile that is, the contents are lost when the power is turned off.

Secondary Memory

Secondary memory is nonvolatile and therefore provides long term storage of programs and data. This kind of storage

Example
* can be magnetic (hard drive)
* optical (CD or DVD)
* Non volatile flash memory (USB)

Input / Output devices

Input / output devices are devices that help us to provide input through devices like mouse and keyboard or output such as a monitor or printer.

Buses

Buses transfer data between each hardware components within a computer system such as between the CPU and main memory.

classification of Hardware Components

The major classification of hardware components are

1. Input devices
2. Processing devices
3. Output devices
4. Storage devices

1. Input devices

The input devices are used to feed data into the computer system. These data is usually provided to some application programs to be processed. The various input devices are Keyboard, Mouse, Joystick, Trackball, Microphone, Scanner, Webcam etc.

2. Processing devices

CPU (Central Processing Unit) is a processing device. CPU is called Brain of the Computer.

A CPU consists of Arithmetic and Logic Unit (ALU), Control Unit (CU) and Memory Unit (MU). ALU performs all arithmetic and logical operations.

Control Unit Controls all the devices attached to the system and overall functioning of the computer.

Memory Unit stores the input data for processing and stores the processed information or future use.

3. Output devices

An output device produces results / output after data processing. The various output devices are Monitor, plotter, Printer, Speakers Projectors etc.

4. Storage devices

Storage devices are used to store information. Temporary data and information is stored in RAM

Permanent information is stored in ROM (Read only Memory).

RAM and ROM are called Permanent memory.

To store large amount of data, secondary memory like Hard disk, Compact disk, Digital Versatile Diskettes, Pendrive etc are used

Computer Software

Computer software is a collection of computer programs that provides instructions to the computer to perform specific tasks.

Two Types of computer software are

1. System Software
2. Application Software

System Software

System Software controls various operations performed by the Computer hardware. It helps the users to operate the Computer software

Examples of System software are Operating system, Assembler, Linker, Loader etc

Application Software

Application software is used to perform operations for a specific task. There are 2 major types of application software.

1. Package

Packages are some general purpose application software developed for users to perform specific tasks based on their requirements. Some of the packages used in computers are.

* Word Processing Software [MS Word]
* Electronic Spread Sheets [MS Excel]
* presentation Graphics software [PPT]
* Database Management Systems (DBMS) [MS Access]

2. Customized Software

Custom software is made as per specific requirements of an organization or user. Also called tailor made software.

Examples: Library Management Systems
Online Ticket Reservation Systems
Hospital Management Systems

Algorithm

An Algorithm is a finite number of clearly described, unambiguous steps that can be systematically followed to produce a desired result for given input in a finite amount of time.

ie) An Algorithm must eventually terminate after completing the set of instructions.

* Computer algorithms are central to Computer science.

* Algorithm provides step by step methods of computation that a machine can carry out.

* A good algorithm gives a program with good performance.

* Algorithms are general computational methods used for solving general problems like determining whether a given number is even or odd.

* Computer can execute instructions very quickly and reliably without error.

An algorithm in general takes a set of values, as input and produces a value or set of values as output.

Algorithm's relation with Programs

Problem

↓

Algorithm

↓

Input → Program → Output

Characteristics of an Algorithm

Input : zero or more input should be supplied to the algorithm

Output : Based on the input, atleast one output is produced as a result.

Definiteness : Every step in the algorithm must be simple and clear. There must not be any ambiguity.

Effectiveness :

Each step must be basic easy to understand and must be able to implement in any programming Language.

Termination

After finite number of steps, every algorithm must have a proper terminating condition.

General characteristics of an Algorithm

* In algorithms every instruction should be precise.
* Every instruction should be unambiguous
* The instructions in an algorithm should not be repeated infinitely.
* An algorithm must eventually terminate
* The algorithm should be written in sequence.
* It looks like normal English.
* The desired result should be obtained only after the algorithm terminates.

Quality of an algorithm

The primary factors that are often used to decide the quality of an algorithm are

* Time requirement
* Memory Requirement
* Accuracy of solution.

Algorithm Analysis

When an algorithm is implemented using a programming language. It uses two main things in a computer.

* The primary memory
* CPU of a computer

* Primary memory is used to hold the data and program.
* CPU is used for the execution of instructions and the program.

Algorithm analysis is analyzing the resource requirements of an algorithm.

* Calculating the time taken for an algorithm to complete and the space requirements it occupies in primary memory is called algorithm analysis.

These processes are called Space Complexity and Time Complexity respectively.

Time Complexity

Time complexity T(n) is defined as the process of determining a formula for total time required to execute that algorithm.

* Time complexity is independent of the programming language, implementation and the programmer.

* It is dependent on the size of the input in that algorithm.

Space Complexity

Space complexity is defined as the process of determining a formula for prediction of how much memory space is required to execute an algorithm.

* This space is usually the primary memory.

Example

Algorithm to find the sum of 2 numbers

[Nov/Dec 2007]

Step 1 : Start

Step 2 : Declare variables A, B and C

Step 3 : Read values A and B

Step 4 : ADD A and B, assign the result to C.

$C \leftarrow A + B$

Step 5 : Print the SUM of 2 numbers, C

Step 6 : Stop

Building Blocks of Algorithms

Some additional features that can be used with the algorithm like control flow that includes looping structures and branching statements.

Functional modules that assist in design of modular programs.

The building blocks of algorithms are

1. Instructions or statements
2. State
3. Control flow
4. Functions

Instructions or Statements

Any single step computation or declaration can be called as a simple instruction or statement in an algorithm.

Example

Algorithm to convert celsius to Fahrenheit

Step 1 : Start

Step 2 : Read the celsius

Step 3 : Calculate the equivalent Fahrenheit using the formula

Fahr ← (1.8 * celsius) + 32

Step 4 : Print the Fahrenheit, Fahr Value

Step 5 : Stop.

State

State defines the current state of the algorithm that includes the variable values, state of execution etc.

State of an algorithm usually is considered in iterative or flow of control statements where there is possibility of changes in the variable values. For example in iterative statements in the jth iteration the value of the variables defined in the algorithm defines the state of the Algorithm.

Example

Algorithm to find the factorial of a number.

Step 1 : Start

Step 2 : Set initial value of factorial f ← 1

Step 3 : Set initial value i ← 1

Step 4 : if i num Goto Step 8

Step 5 : f ← f * i

Step 6 : i ← i + 1

Step 7 : Goto Step 4

Step 8 : print f

Here in each iteration the value i and f changes based on the computational steps defined in the algorithm.

In iteration 3, i has a value 3 f has the value 6. This defines a state of the algorithm.

Control flow

In case of some logical problems, simple instructions are not enough to derived the results.

* In that case we need some logical computations and based on the runtime values generated, decision must be taken and steps should be proceeded.
* The control flow statements like if-then, if-then-else, if-then-elif and looping instructions like for while can be used for iterative computations

Algorithm

To check whether a number of divisible by 3 or not

Step 1 : Start

Step 2 : Read the input number, n

Step 3 : If (n %. 3) = 0 Then

print "The number is divisible by 3

else

print "The number is not divisible by 3

Step 4 : Stop

Functions

Repeated set of statements can be grouped and defined as functions. They can be called in our main flow of the algorithm as and when needed.

* In the sum of series problems, we can use define factorial() as a seperate function to find the factorial of a number in the series.
* Problems of larger sizes and too many computations uses functions to make the task modular and readable.

Example

Algorithm to print the sum of series

$$\frac{1}{1!} + \frac{2}{2!} + \frac{3}{3!} + \cdots \frac{n}{n!}$$

The main function can be defined to calculate the sum of the results.

* A subroutine or function can be used to find the factorial of the denominators one by one.

Algorithm for main function

Step 1 : Start

Step 2 : Read the value of n

Step 3 : Assign Sum ← 0, term 1, i

Step 4 : term ← n / Factorial (n)

Step 5 : Sum ← Sum + term

Step 6 : i ← i + 1

Step 7 : if i <= n Goto step 4

Step 8 : print the Sum

Step 9 : Stop

Example

Algorithm for factorial function

Step 1 : Start

Step 2 : Set initial value of factorial f ← 1

Step 3 : set initial value i ← 1

Step 4 : if i > num Goto step 8

Step 5 : f ← f * i

Step 6 : i ← i + 1

Step 7 : Goto Step 4

Step 8 : Return f

Heuristic Algorithms

When solving any computational problem, any suitable existing algorithm can be used or a new algorithm can be developed.

* There are so many standard algorithms for doing mathematical computations, logical problems like choosing a best move in chess game, shortest path algorithms and so on

Even then, there are some algorithm that work well in general but are not guaranteed to give correct result for each specific problem.

Such algorithms are called Heuristic Algorithms

Example: Traveling Salesman problem
Searching
Sorting

Notations used in Problem Solving

Basic notations used to describe algorithms are

1. Pseudo code
2. Flowchart
3. Programming Languages

Pseudocode

Pseudo code (derived from pseudo and code) is a notation that is used to develop the solution of a problem.

Pseudo means imitation and code refers to the instructions written in a programming language.

* Pseudocode is hence, an imitation of actual computer instructions.
* It uses the structural conventions of programming languages, but omits detailed subroutines or language specific syntax.
* A Pseudo code is not a program It can be used to generate a program using any programming Language

Note: Data declarations must not be included in pseudocode.

Pseudo code to print pass or fail based on the grade.

```
IF student's grade is greater than or
                              equal to 60
Print " passed"
else
    print " Failed"
Endif
```

Pseudo code to sort an array of number

```
Set n to number of records to be sorted
repeat
Flag = False;
for counter = 1 to n-1 do
if key[counter] > key[counter+1] then
swap the records;
Set flag = true;
end if
end do
n = n-1;
until flag = false or n=1
```

Pseudocode for printing first 20 numbers in Fibonacci series

```
Initialise n to 20
Initialise sum to zero
Initialise f1 and f2 to zero
repeat n times
add f1 and f2, Store this value in sum
assign f1's current value to f2
assign sum's current value to f1
end loop
```

Flowchart

Flowchart is a diagrammatic or pictorial representation of an algorithm, that defines the steps in finding solution to a problem.

* An algorithm serves as a guideline or roadmap for the programmer to know the flow of execution steps.

Flowchart conventions and rules

1) only the predefined shapes must be used to construct the flowchart. we must not create new shapes.
2) The flow of the program is usually from top to bottom, the directed arrow heads are used to represent the flow of logic.
3) For a process symbol there is only one incoming arrow and one outgoing arrow head.
4) Only for decision symbol, there will be two outgoing arrow heads for true and false conditions.

5) The flow lines should not cross each other.

6) Be consistent in using names and variables in the flowchart.

7) keep the flowchart as simple as possible. Don't chart every details or else the flowchart will only be a graphical representation

8) Words in the flowchart symbols should be common statements and easy to understand.

9) Chart main line of logic and then incorporate all the details of logic

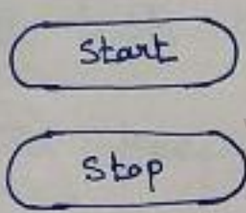
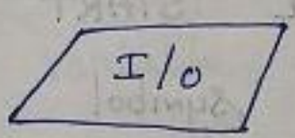
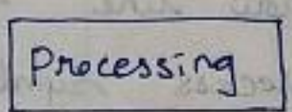
10) If a new page is needed for flowcharting, then use connectors for better representation.

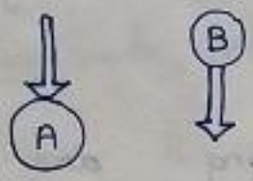
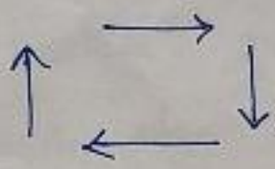
Benefits of using flowchart in problem solving

* problem logic can be easily understood.
* Helps to analyze the problem precisely.

* Helps to generate efficient coding and documentation
* Aids in systematic testing and debugging.
* Helps in program maintenance.

Flow chart Symbols

Symbols	Purposes
	They are called terminal symbols. They are used to define the begin and end of a flowchart.
	Parallelogram to get input and print results.
	A rectangle shape for computational instructions variable initializations
	Decision making leads to branches based on some condition checking. There are two exits only for this shape.

Symbols	Purposes
	connectors are used when the flowchart is complex with more than 1 page or overlapping flows or crossover. Simpler constructs can be formed with connectors.
	Arrow to connect each symbol. These arrow heads denote the flow of control in the program.

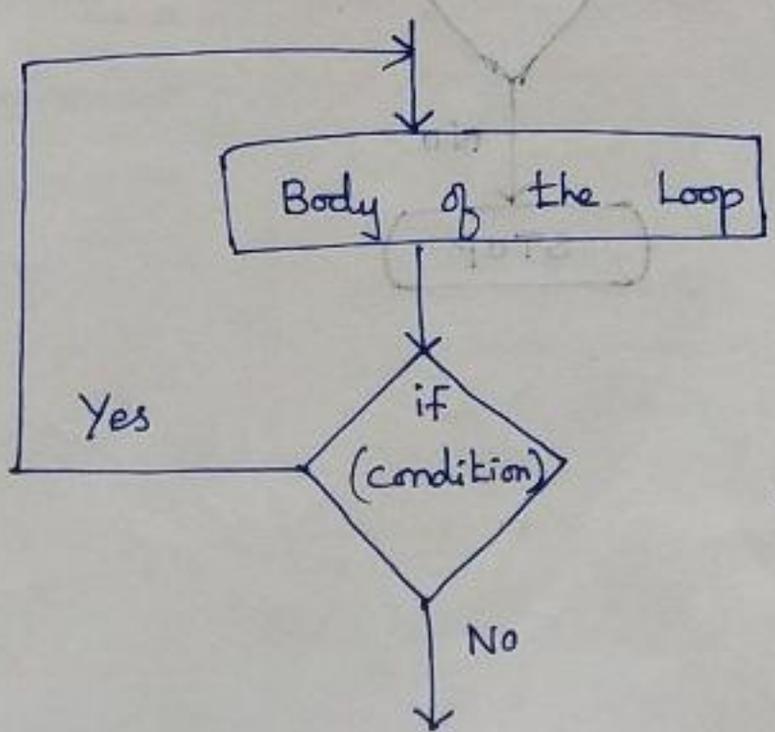
Features of a flowchart

* Every flowchart has a START symbol and a STOP symbol
* There is only one flow line that comes out from a process symbol.
* The general direction of flow in any flowchart is from top to bottom or left to right.
* Only one flow line is used to connect the START and STOP symbols with other shapes of the flowchart.

* A decision box has only one entry point but 2 exit points. one exit point is for TRUE condition and other exit point is for FALSE condition

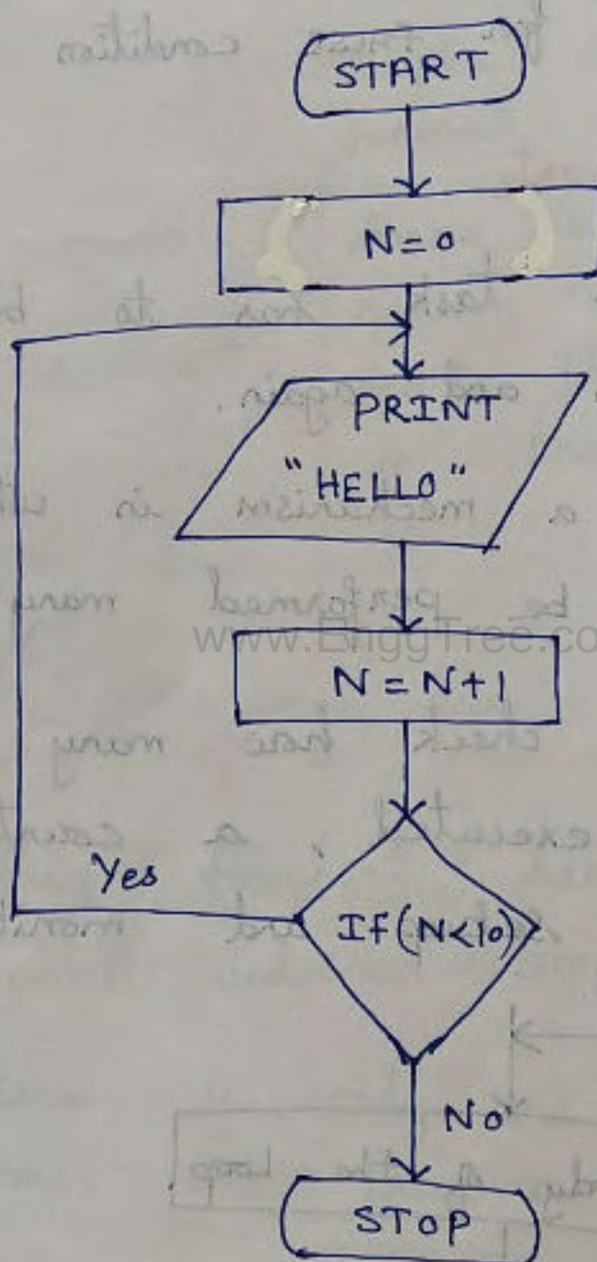
Loops in flowcharts

* Sometimes a task has to be repeated again and again.
* The loop is a mechanism in which a task can be performed many times.
* To keep a check how many times a loop is executed, a counter variable is setup and monitored.



Example

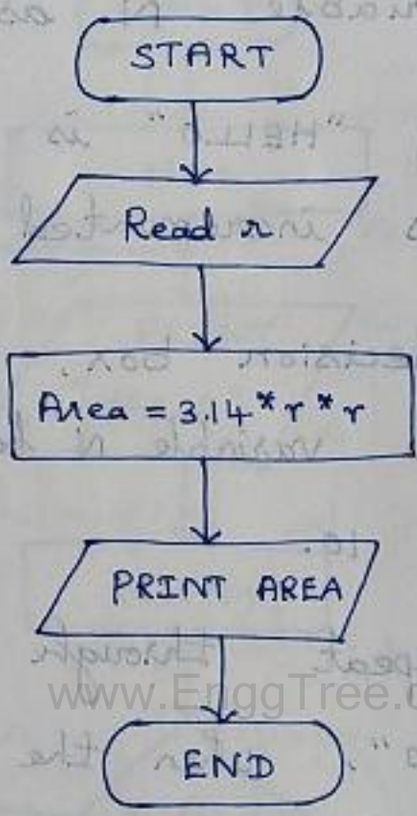
A flowchart to print "HELLO" ten times



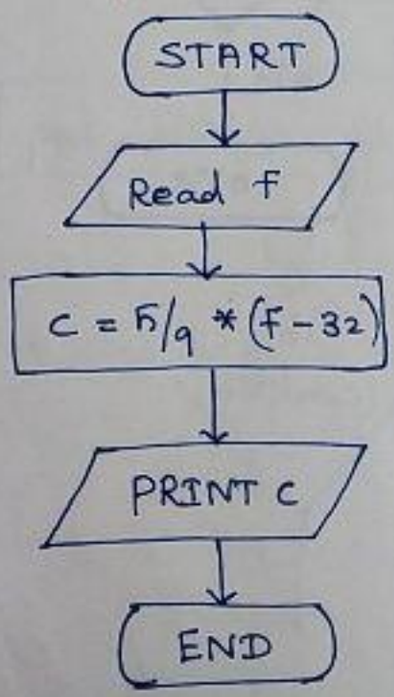
* Here Start and stop defines the begin and end of the flowchart.
* A process box declares the counter variable N as 0.
* The String "HELLO" is printed and N is incremented by 1.
* In the decision box, check whether the counter variable N has reached the value 10.
* If not repeat through the step print "HELLO", when the counter variable reaches 10, the flowchart is terminated.

Example:

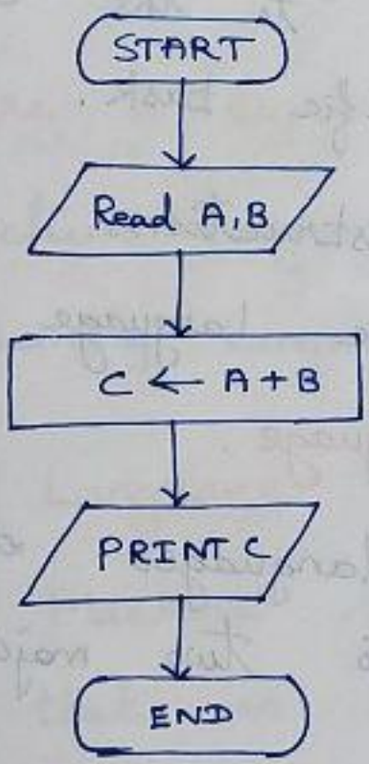
Find the area of a circle of radius r



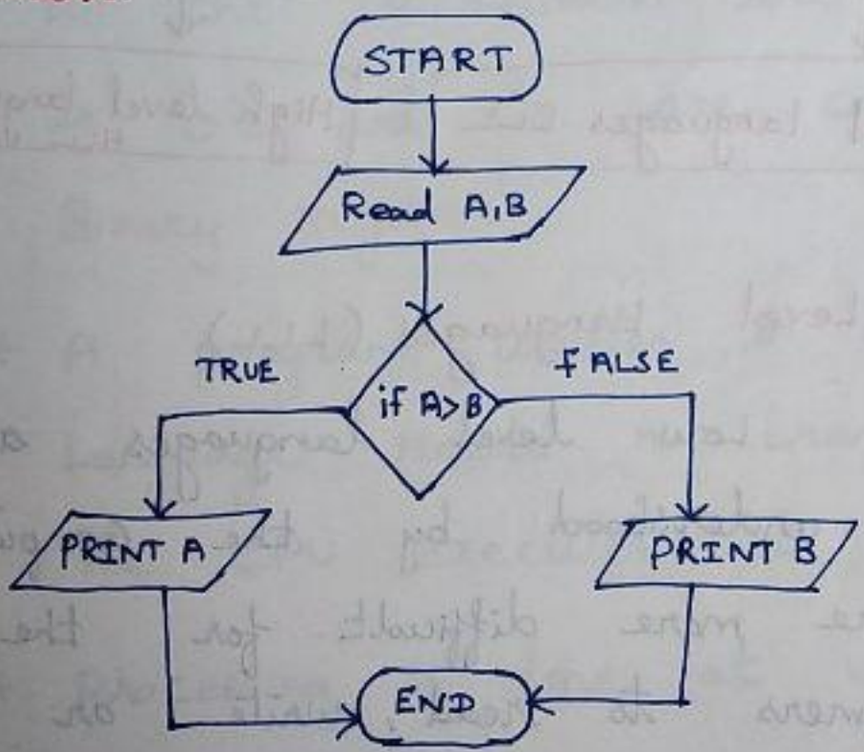
Convert temperature Fahrenhit to celsius



Get two numbers and prints sum of their value.



Find the greater number between two numbers.

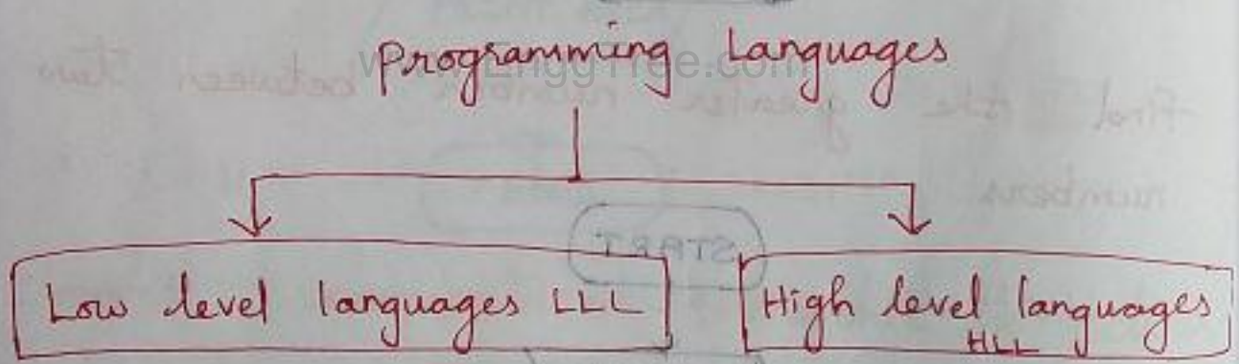


Programming Languages

* A program is a set of instructions given to the computer to perform a specific task.

* These instructions are written in some specific Language called Programming language.

* Programming languages are generally categorized into two major types



1. Low Level Languages (LLL)

Low level languages are easily understood by the computer, but are more difficult for the programmers to read, write or understand the machine Language.

* Low level language is a type of Programming Language.

There are 2 types of Low Level Language

1. Machine Language
2. Assembly Language

Machine Language

Machine language is the only language that is directly understood by the computer.

* All the instructions are written in 0's and 1's are called Binary Digits.
* A program written in a machine Language needs no translation The CPU executes it directly.
* Processing is done at very fast speed when programs are written using machine language.

Limitations of Machine Language

Inspite of good performance, machine languages are not widely used. Few reasons are listed

* Machine languages are not easy to understand by human beings.
* Writing a machine language is very difficult, and can be done only by expert programmers.
* Consumes more time for writing the program as well as for modification and debugging.
* The programmer has to remember all the codes with 0's and 1's

Assembly Language

* Assembly Language uses mnemonic codes instead of binary digits.
* Assembly Language use keywords and symbols much like English, to form a programming Language.

* The computer doesn't understand the assembly code, so we need a way to convert it to machine code, which the computer does understand.

* Assembly Language programs are translated into machine language program by an assembler

List of operations and mnemonics

Operation	Mnemonic
Addition	ADD
Subtraction	SUB
Multiply	MUL
Load	LDA
Divide	DIV
Store	STA
Jump	JMP
Terminate	HALT
Increment	INR

These mnemonic codes are unique to a processor and the instruction set. The programmer must have a good knowledge of the mnemonics of that processor and its format and purpose to write effective Programs.

High Level Languages (HLL)

* This type of programs are easy for the programmers to write and test.
* These languages are written with letters, words and mathematic symbols that can be easily understood by the humans.
* High level language programs are like human languages
* High level languages are not machine dependent.
* Some of the high level languages are C, C++, Java
* Compared to machine languages and assembly languages, high level languages are easy to write, debug and maintain.
* But to run high level language program it must be converted to machine code before executing. This conversion is done by compiler or assembler.

Compiler is a software program, that translates the source program written in high language into its equivalent target program in low level language.

Language Processors

A Language Processor is also known as Language Translator.

* It is a software used to convert the high level language codes into machine language codes.
* When a program is fed into the computer, the language translator automatically converts the program into machine language and check its syntax for any errors.
* There are 3 types of language translators

1. Assembler
2. Interpreter
3. Compiler.

1. Assembler

An assembler is a software that converts the assembly language instructions into the machine codes.

2. Interpreter

An interpreter is a software that converts the high level program codes into machine language.

* It translates the high level language program line by line.
* A line is translated, executed and then it moves to the next line.
* If an error is found on any line, it is reported to the user and the execution of the program is stopped.

3. Compiler

A compiler also converts the high level language program into machine language.

* It converts the complete high level language program into machine code
* If any error is found, the whole program has to be compiled again.

Algorithmic Problem Solving

Program 1

To perform Multiplication of 2 numbers like 12 and 20

Algorithm

Step 1 : Start

Step 2 : Declare variables A, B and C

Step 3 : Read values A and B

Step 4 : Multiply A and B, assign the result to C.

C ← A * B

Step 5 : Print the product of 2 numbers, C

Step 6 : Stop.

Pseudocode

READ the value of 2 numbers

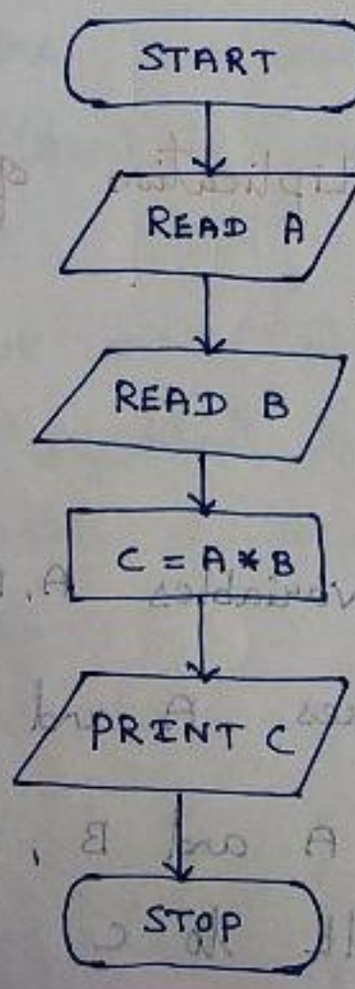
Find the product of two numbers using the statement

C = A * B

WRITE the output to print the product of 2 values.

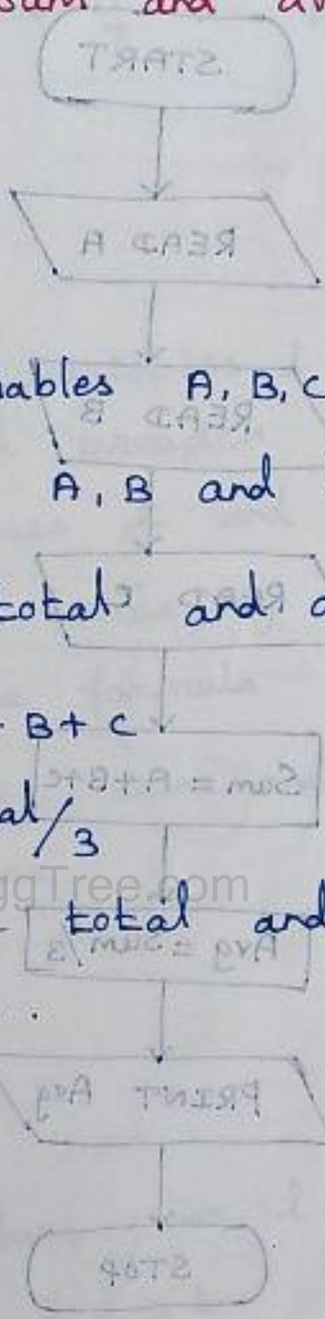
STOP

Flowchart



Program 2

To compute the sum and average of 3 numbers



Algorithm

Step 1 : Start

Step 2 : Declare Variables A, B, C

Step 3 : Read values A, B and C

Step 4 : Find the total and avg

total = A + B + C

Avg = total / 3

Step 5 : print the total and Avg of the 3 numbers.

Step 6 : Stop.

Pseudocode

READ the values for A, B and C

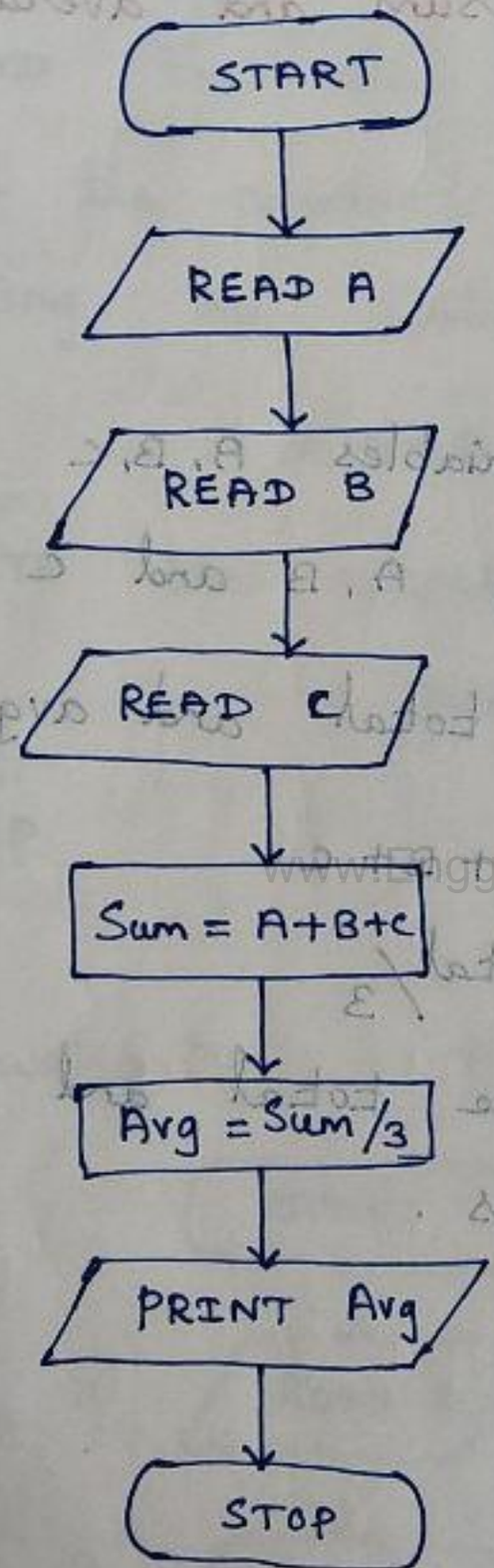
Calculate total = A + B + C

Calculate Avg = total / 3

WRITE the values of total, Avg

STOP

Flowchart



Program 3

To compute Area of a Rectangle

Algorithm

~~Start~~ :

Step 1 : Start

Step 2 : Declare variables l and b for length and breadth

Step 3 : Read values l and b

Step 4 : Find the area of rectangle using the formula Area = l * b

Step 5 : Print the sum and Avg of the 3 numbers.

Step 6 : Stop

Pseudocode

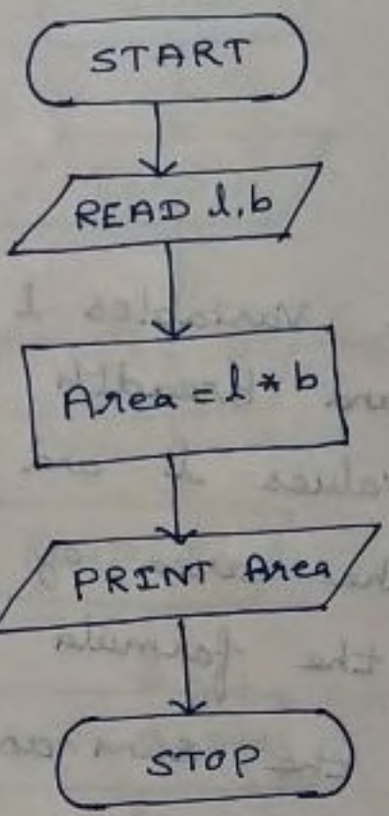
READ the values for l and b

calculate Area = l * b

WRITE the Area of Rectangle. Area

STOP

Flowchart



Program 4 :

To find the cube of a number

Algorithm

Step 1 : Start

Step 2 : Read a number a

Step 3 : Find the cube of the number using the formula

Cube = a * a * a

Step 4 : Print the cube of the value

Step 5 : Stop

Pseudocode

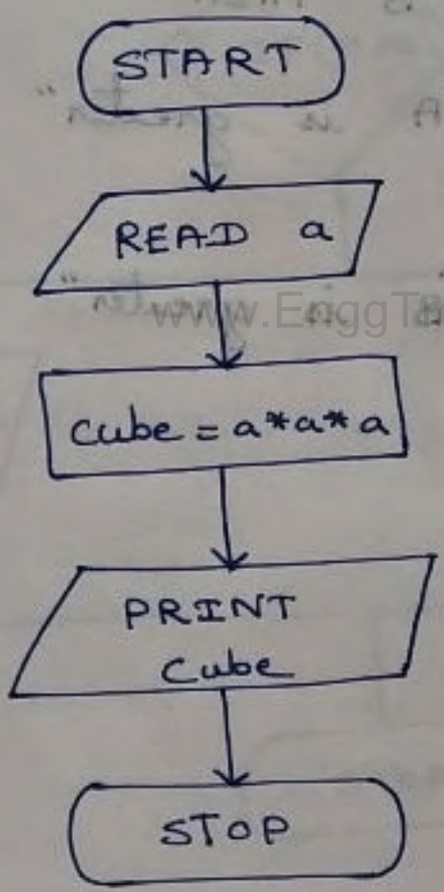
READ the values a

calculate cube = a * a * a

WRITE Cube of the number, cube

STOP

Flowchart



Program 5

To find the greater of 2 numbers

Algorithm

Step 1 : Start

Step 2 : Read 2 numbers A and B

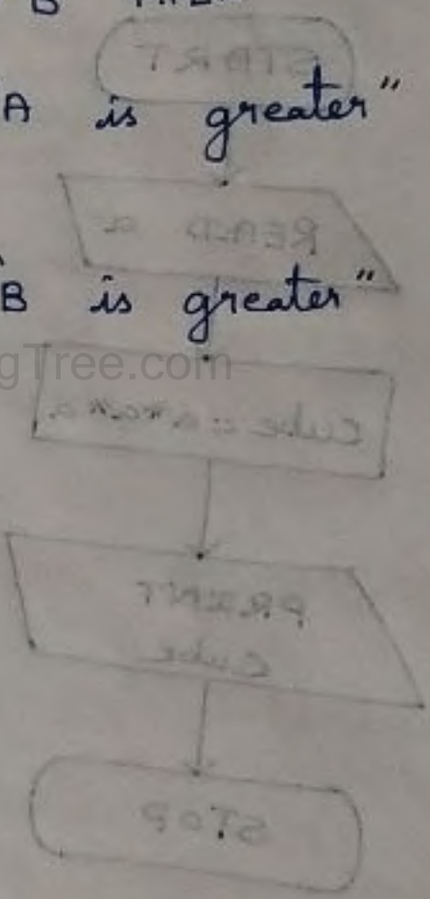
Step 3 : IF A > B THEN

Print "A is greater"

ELSE

Print "B is greater"

Step 4 : Stop

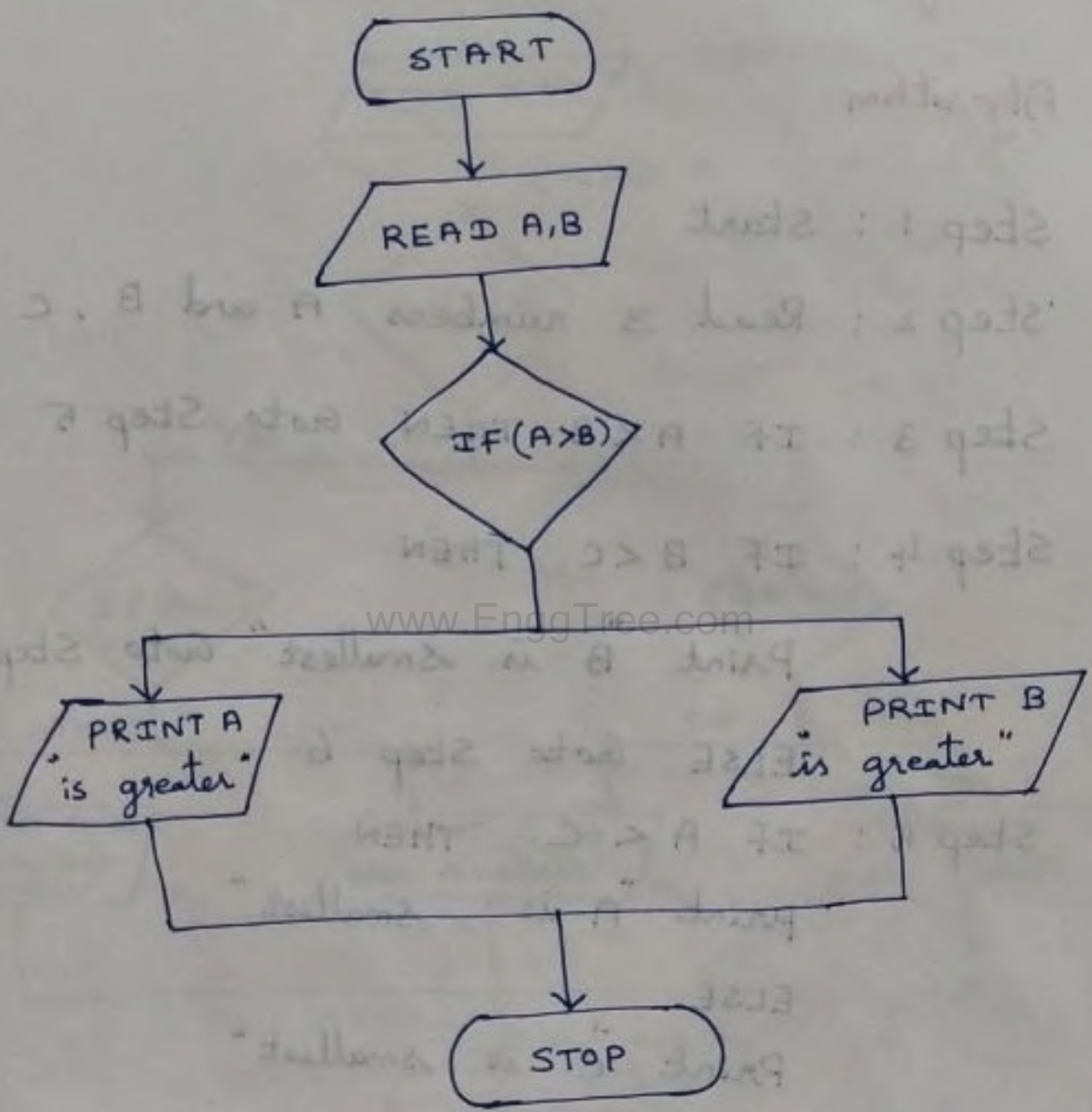


Pseudocode

```
READ   A, B
IF (A>B)
    write "A is greater"
ELSE
    Write "B is greater"
STOP
```

Program

Flowchart



Program - 6

To find the smallest of three numbers

Algorithm

Step 1 : Start

Step 2 : Read 3 numbers A and B , C

Step 3 : IF A < B THEN Goto Step 5

Step 4 : IF B < C THEN

Print "B is smallest" Goto Step 6

ELSE Goto Step 6

Step 5 : IF A < C THEN

print "A is smallest"

ELSE

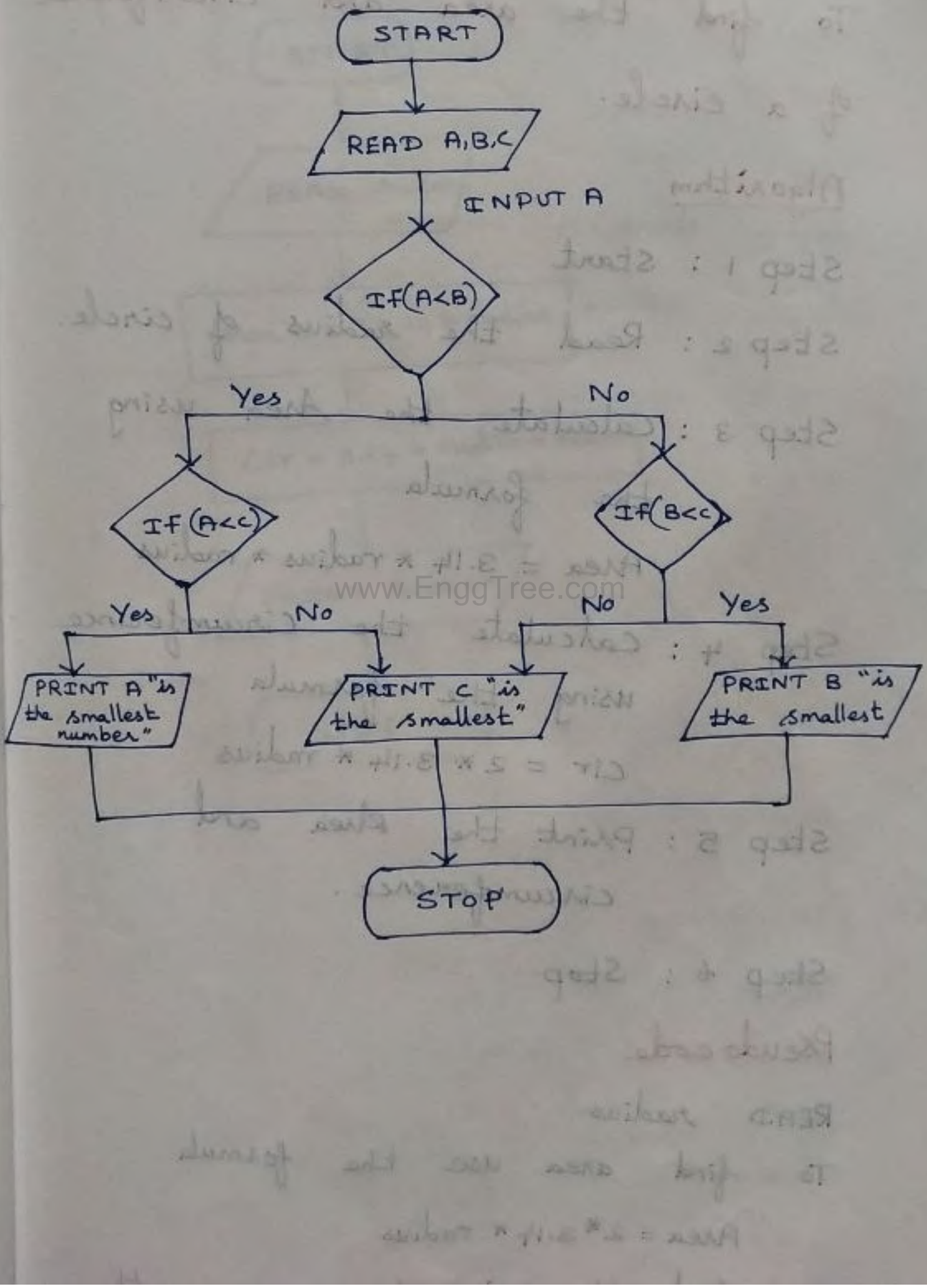
Print "C is smallest"

Step 6 : Stop

Pseudocode

```
READ the values of A, B and C
If (A < B)
    If (A < C)
    Write "A is smallest"
    ELSE Write "C is smallest"
ELSE
    If (B < C)
    Write "B is smallest"
    ELSE Write "C is smallest"
STOP
```

Flowchart



Program 1

To find the area and circumference of a circle.

Algorithm

Step 1 : Start

Step 2 : Read the radius of circle.

Step 3 : Calculate the Area using the formula

Area = 3.14 * radius * radius

Step 4 : Calculate the circumference using the formula

Cir = 2 * 3.14 * radius

Step 5 : Print the Area and circumference.

Step 6 : Stop

Pseudo code

READ radius

To find area use the formula

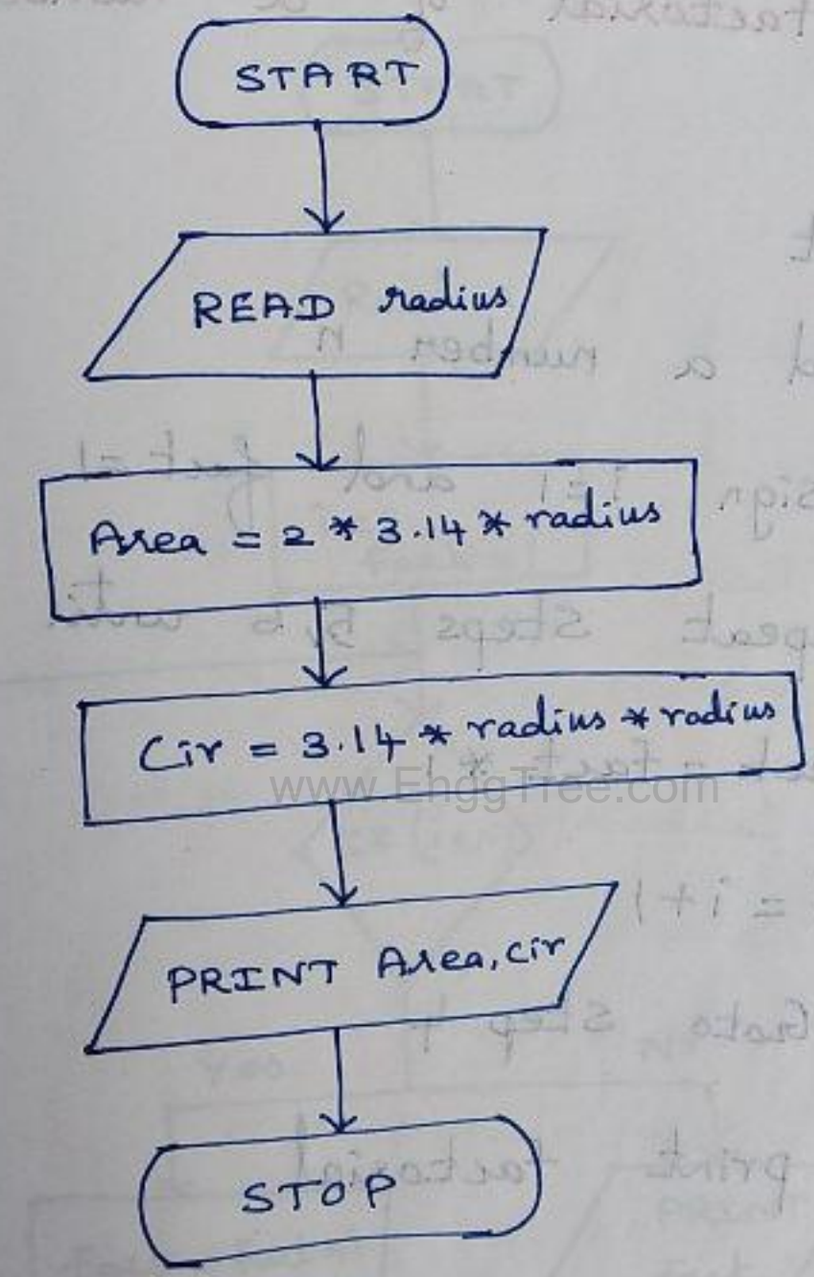
Area = 2 * 3.14 * radius

To find the circumference use the formula Cir = 3.14 * radius * radius

Write Area and Cir

STOP

Flowchart



Program : 8

To Find Factorial of a number

Algorithm

Step 1 : Start

Step 2 : Read a number n

Step 3 : Assign i=1 and fact=1

Step 4 : Repeat steps 5, 6 until i<=n

Step 5 : fact = fact * i

i = i+1

Step 6 : Goto step 4

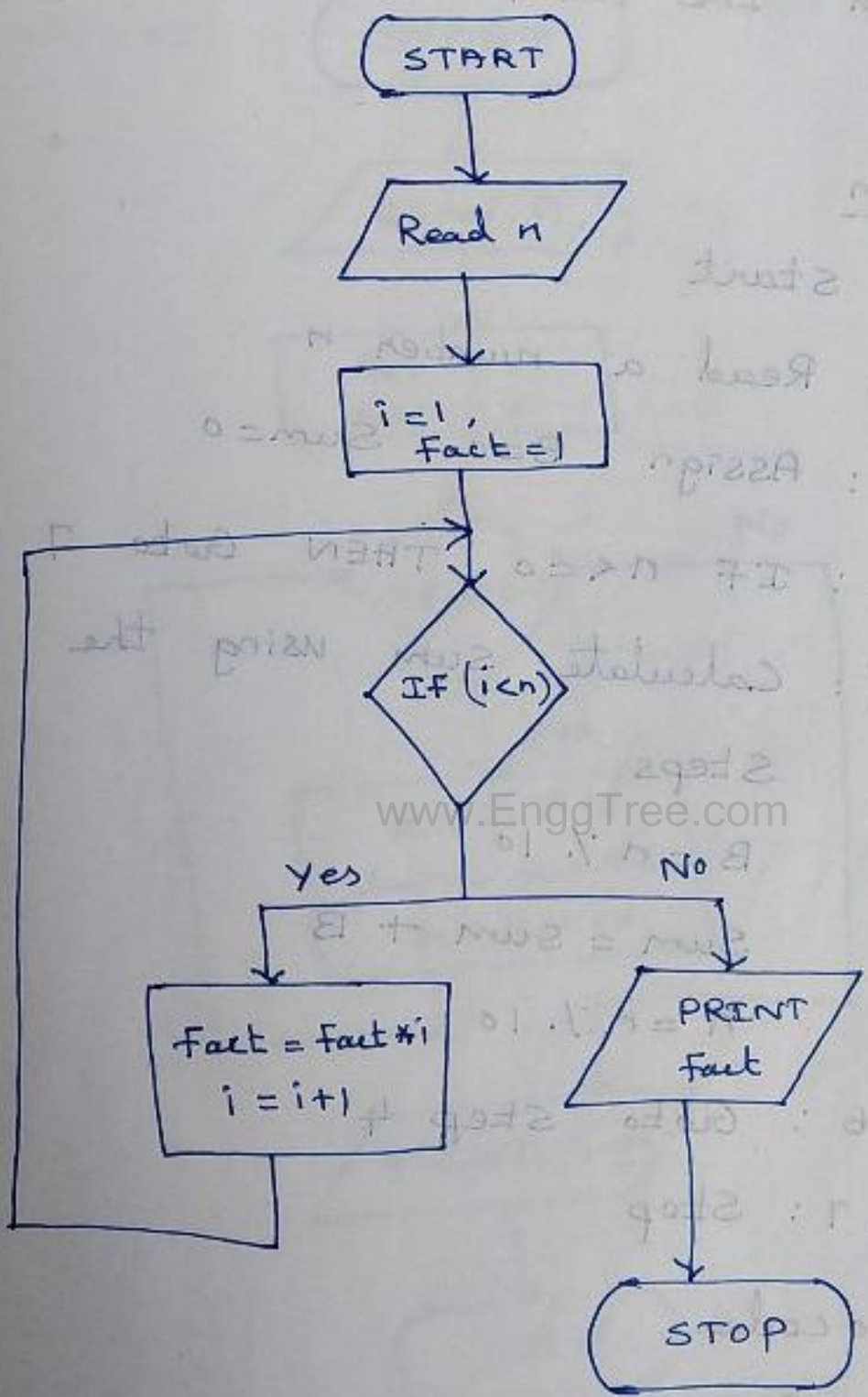
Step 7 : print factorial

Step 8 : Stop

Pseudocode

```
READ  N
Set  i ← 1 , Fact ← 1
WHILE (i<=N)
fact = Fact * i
i = i+1
Repeat until the condition fails
WRITE Fact
STOP
```

Flowchart



Program 9

To find the sum of digits of a number

Algorithm

Step 1 : Start

Step 2 : Read a number n

Step 3 : Assign B=0 Sum=0

Step 4 : IF n<=0 THEN Goto 7

Step 5 : Calculate sum using the Steps

B = n % 10

Sum = sum + B

n = n % 10

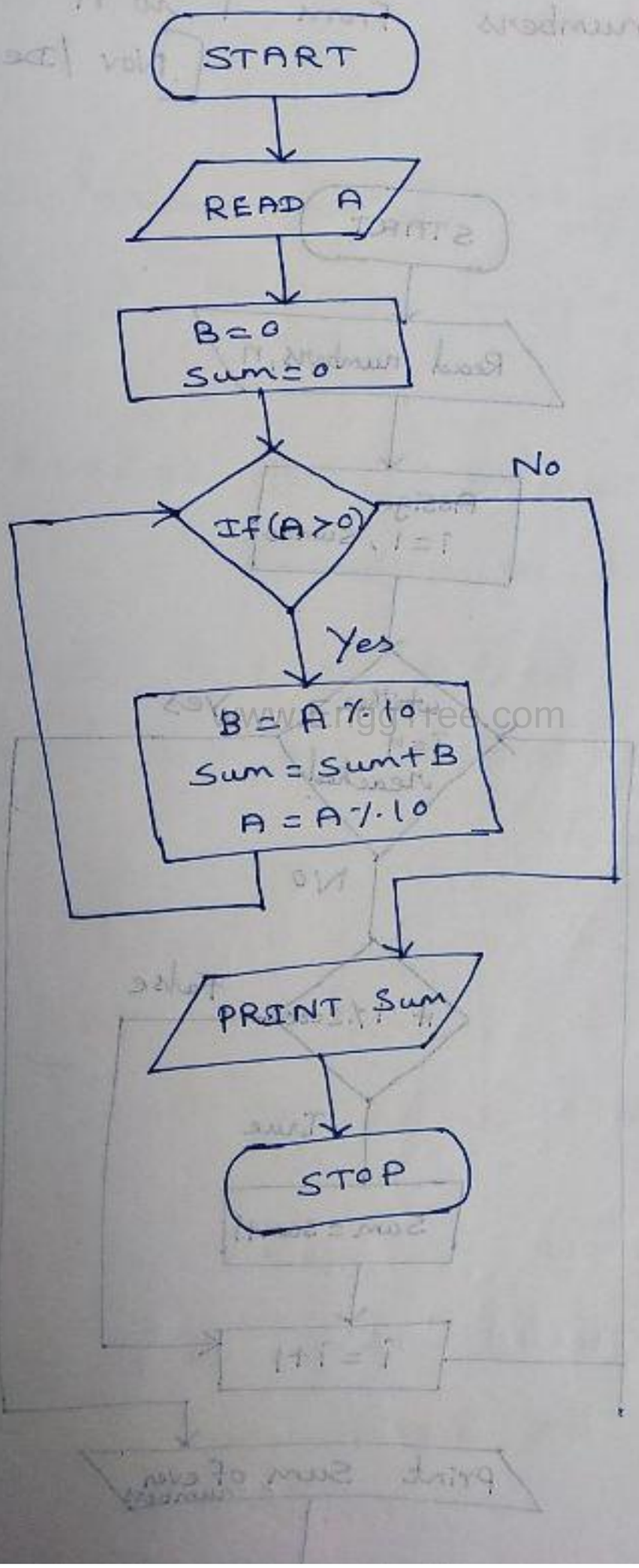
Step 6 : Goto step 4

Step 7 : Stop

Pseudo code

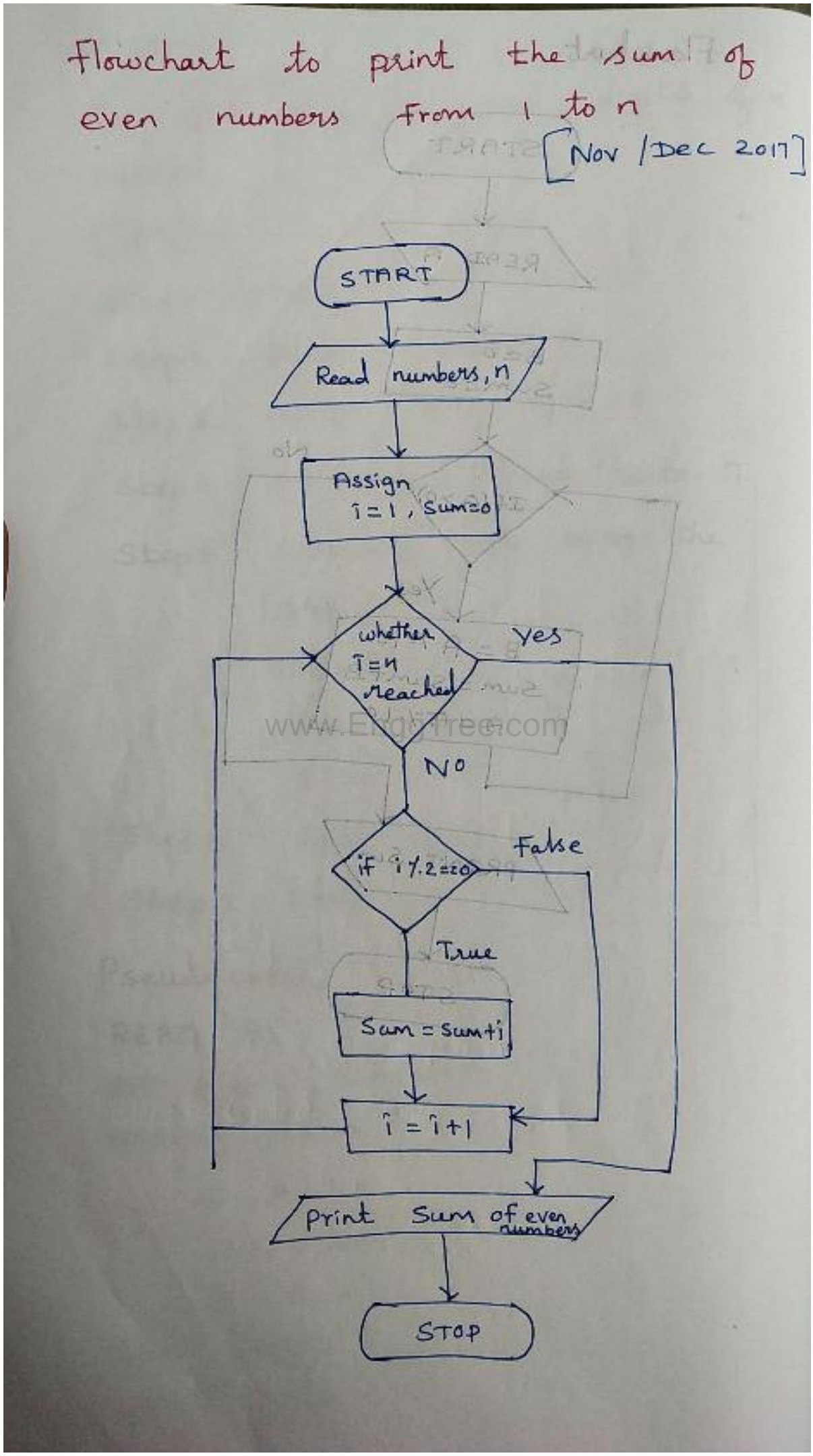
```
READ A
SET B ← 0   Sum ← 0
WHILE (A>=0)
    B = A % 10
    Sum = sum + B
    A = A / 10
END WHILE
WRITE Sum
STOP
```

Flowchart



Flowchart to print the sum of even numbers from 1 to n

[Nov /Dec 2017]



Simple strategies for Developing Algorithms

Iteration

An iterative control statement allows repeated execution of set of statements.

* An iterative control statement is a set of instruction
* The iterative control statement that controls the execution of the statements.
* Because of the repeated execution iterative control structures are commonly called as "loops"
* Iterations in Python is best implemented using "while" loops and "For loops"

Syntax

```
while condition :
    Body of while
```

* With the while loop we can execute a set of statements as long as a condition is true.

Example

```
i = 1
while i < 6 :
    print (i)
    i += 1
```


output

```
1
2
3
4
5
```

Note

Remember to increment i or else the loop will continue forever

The break Statement

With the break statement we can stop the loop even if the while condition is true

Example

Exit the loop when i is 3

```
while  i < 6 :
    print (i)
    if  i == 3:
        break
    i += 1
```

Output

```
1
2
3
```

The Continue statement

With the continue statement we can stop the current iteration and continue with the next.

Example

Continue to the next iteration if i is 3

```
i = 0
while  i < 6 :
    i += 1
    if i == 3 :
        continue
    print (i)
```

output

```
1
2
4
5
6
```

For loops

* A for loop is used for iterating over a sequence

* With the for loop we can execute a set of statements once for each item in a list, tuple, set etc.

Example

Print each fruit in a fruit list

```
Fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

output

```
apple
banana
cherry
```

Looping through a String

String contain a sequence of characters

Example

Loop through the letters in the word "apple"

```
For x in "apple":
    print (x)
```

output

```
a
p
p
l
e
```

The break Statement

With the break statement we can stop the loop before it has looped through all the items

Example

Exit the loop when x is barara

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
    if x == "banana":
        break
```

output

```
apple
banana
```

Example

Exit the loop when x is "banana" but this time the break comes before the print

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

Output

apple

The continue statement

With the continue statement we can stop the current iteration of the loop and continue with the next:

Example

Do not print banana

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

Output

apple
cherry

The range() Function

To loop through a set of code a specified number of times, we can use the range() function

The range() function returns a sequence of numbers, starting from 0 by default and increments by 1 by default and ends at a specified number.

Example

Using the range() function

```
for x in range(6):
  print(x).
```

Output

```
0
1
2
3
4
5
```

The range()

The range() function defaults to increment the sequence by 1.

* It is possible to specify the increment value by adding a third parameter range(2, 30, 3):

Example

Increment the sequence with 3

```
for x in range(2, 30, 3):
    print(x)
```


output

```
2
5
8
11
14
17
20
23
26
29
```

Else in for loop

Example

print all numbers from 0 to 5, and print a message when the loop has ended.

```
for x in range(6):
    print(x)
else:
    print ("Finally finished")
```

Output


```
0
1
2
3
4
5
Finally finished
```

Note:

The else block will not be executed if the loop is stopped by a break statement.

Example

```
for x in range(6):
    if x==3: break
    print(x)
else:
    print("finished")
```

output

```
0
1
2
```


Nested Loops

A nested loop is a loop inside a loop.

The inner loop will be executed one time for each iteration of the outer loop

Example

```
a = ["red", "big", "tiger"]
b = ["apple", "cherry", "orange"]
for x in a :
    for y in b :
        print (a,b)
```

Output

```
red   apple
red   cherry
red   orange
big   apple
big   cherry
big   orange
tiger apple
tiger cherry
tiger orange
```

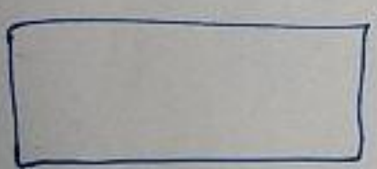
The Pass Statement

* for loops cannot be empty
* Some reason for loop with no content
* put in the pass statement to avoid getting an error.

Example

```
For x in [0, 1, 2]:
    pass
```

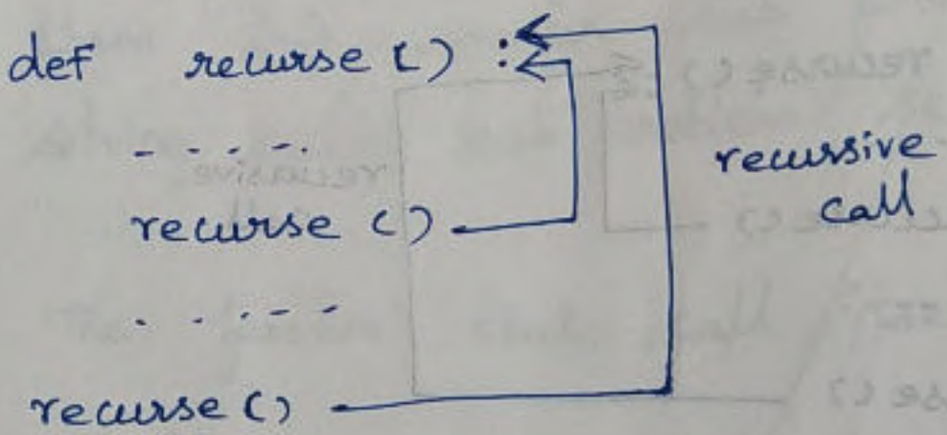

output



Empty screen, no error.

Recursion

* The function that call itself



Example

```
def Factorial (n):
    if (n==1) :
        return 1
    else
        return  n * factorial (n-1)
n = int (input (" Enter  n value"))
res = factorial (n)
print (res)
```

Iteration

```
n = int(input ("Enter number :"))
fact =1
while (n>0):
    fact = fact * n
    n =n-1
print("Factorial of the number is : ")
print (fact)
```

output

```
Enter number : 5
Factorial of the number is :
120
```

ILLUSTRATIVE PROBLEMS

Program : 1

Find minimum elements in list of items

Algorithm

1. Set min to arr[0]
2. Start loop at 1 using index i = 1
3. Check if arr[i] < min, if so set min to arr[i]
4. Check if arr[i] > max, if so set max to arr[i]
5. Repeat through step 3, till i becomes n
6. Print the minimum item.

Program

```
list1 = [123, 'xyz', 'Game', 'abc']
list2 = [456, 700, 200]
print "Min value element:", min(list1)
min value element : 123
print "Max value element:", max(list2)
max value element : 700
```

output

```
Min value element : 123
Max value element : 700
```

Note: Comma helps to write more statements in a single line.

Program : 2

Insert a card in a list of sorted cards

Algorithm

Step 1 : start

Step 2 : import bisect function

Step 3 : create the list

Step 4 : insort function used insert the number and get sorted list.

Step 5 : print the sorted list.

Program

```
import bisect
a = [11, 22, 33, 44, 66, 88]
bisect.insort(a, 77)
print a
```


Output

[11, 22, 33, 44, 66, 77, 88]

bisect → This function returns the position in the sorted list

insort → This function returns the sorted list after inserting number in appropriate position.

Jan 2019 - 8 Mark
Jan 2018 - 16 Mark

Program : 3

Tower of Hanoi problem

Problem Statement

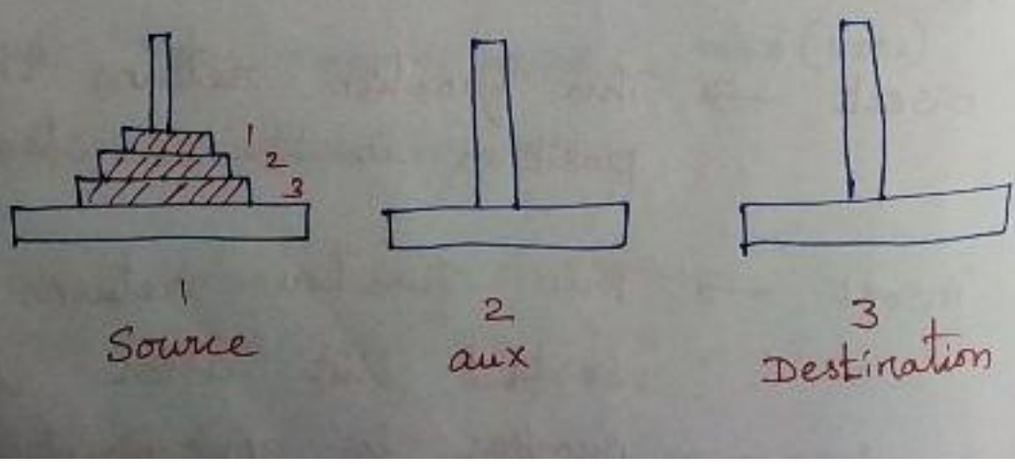
Tower of Hanoi is one of the classical problems of computer science.

The problem states that.

1. There are three stands (Stand 1,2,3) on which a set of disks, each with a different diameter are placed

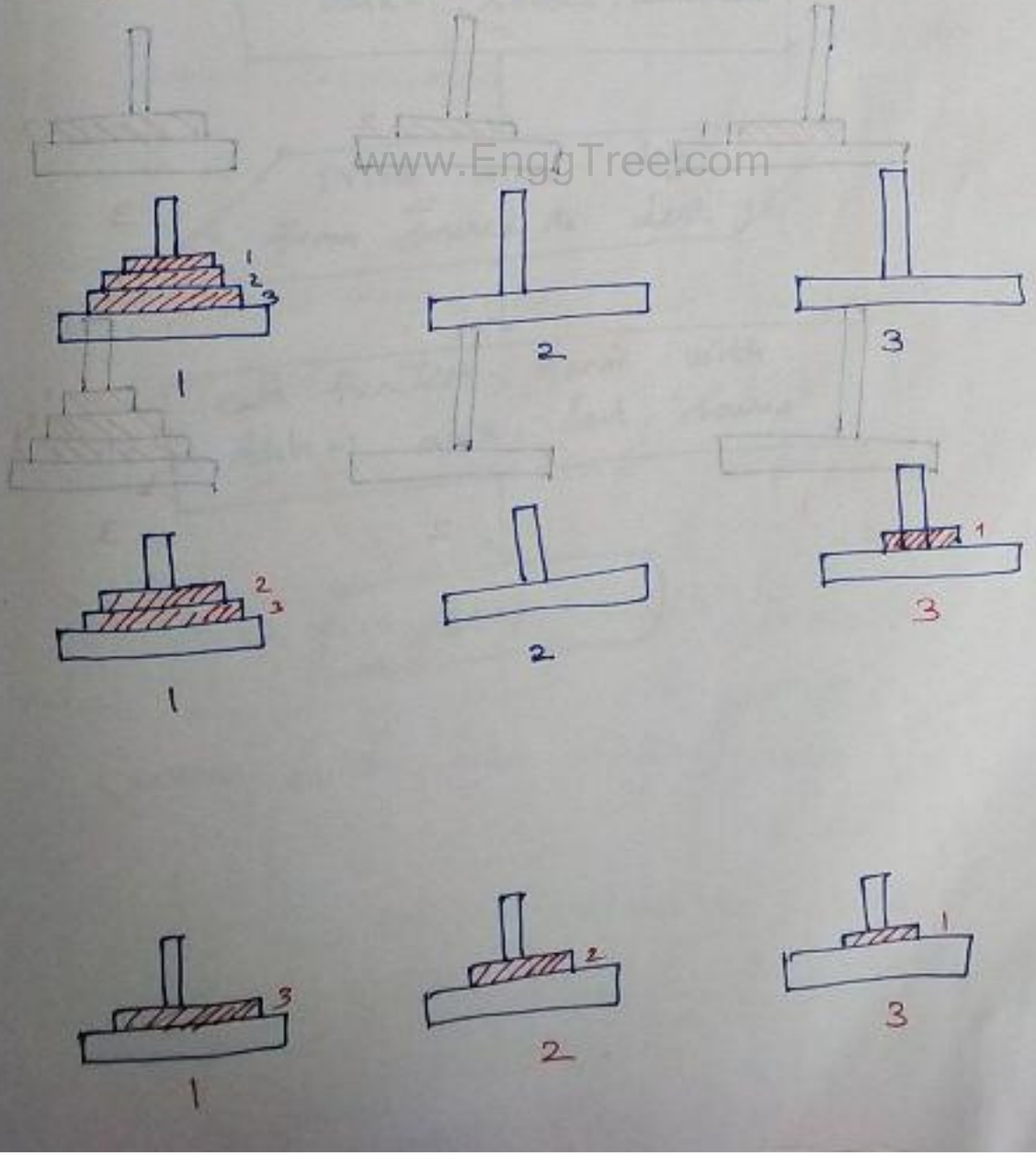
2. Initially the disk are stacked on stand 1, in order of size, with the largest disk at the bottom.

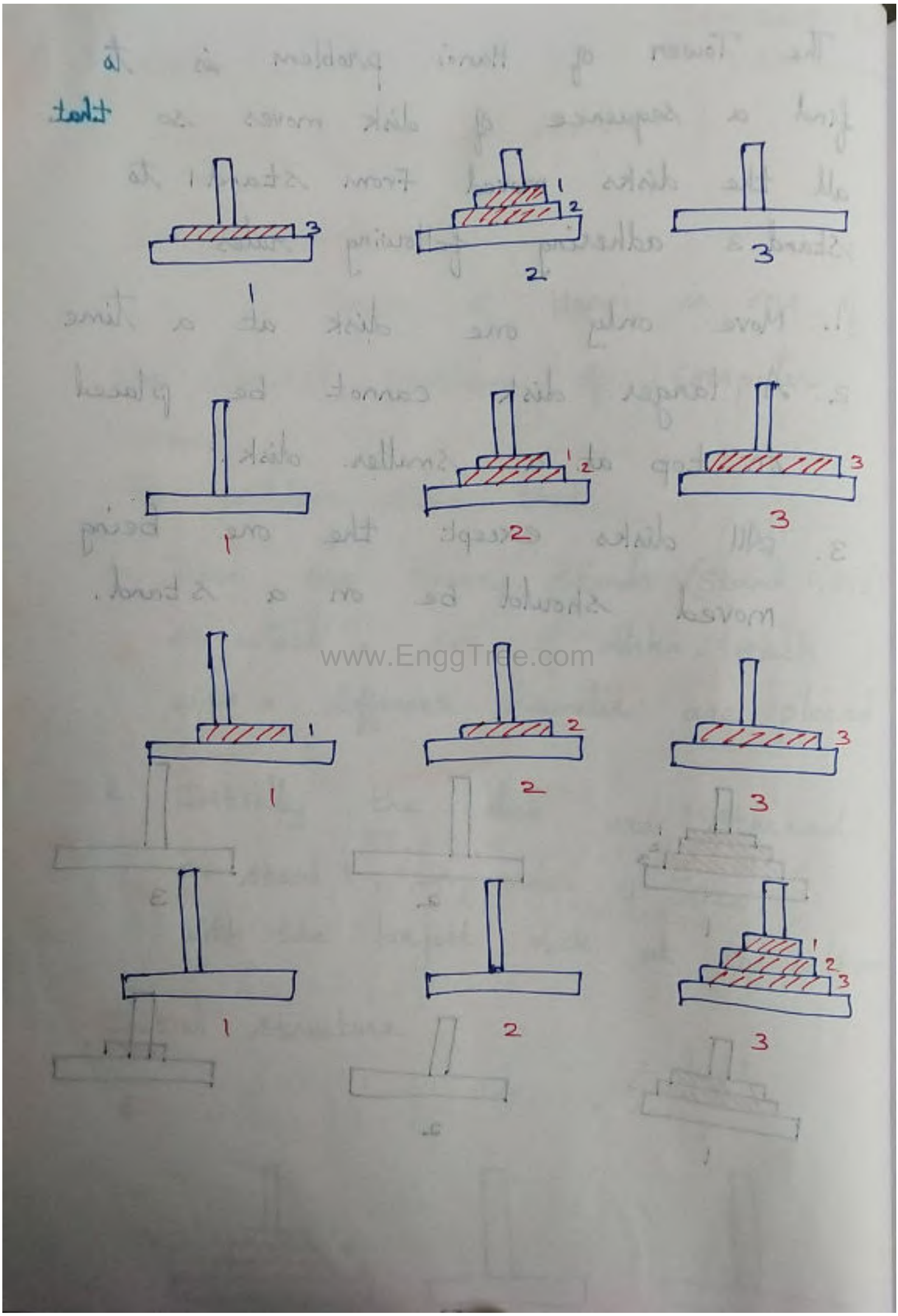
Initial structure



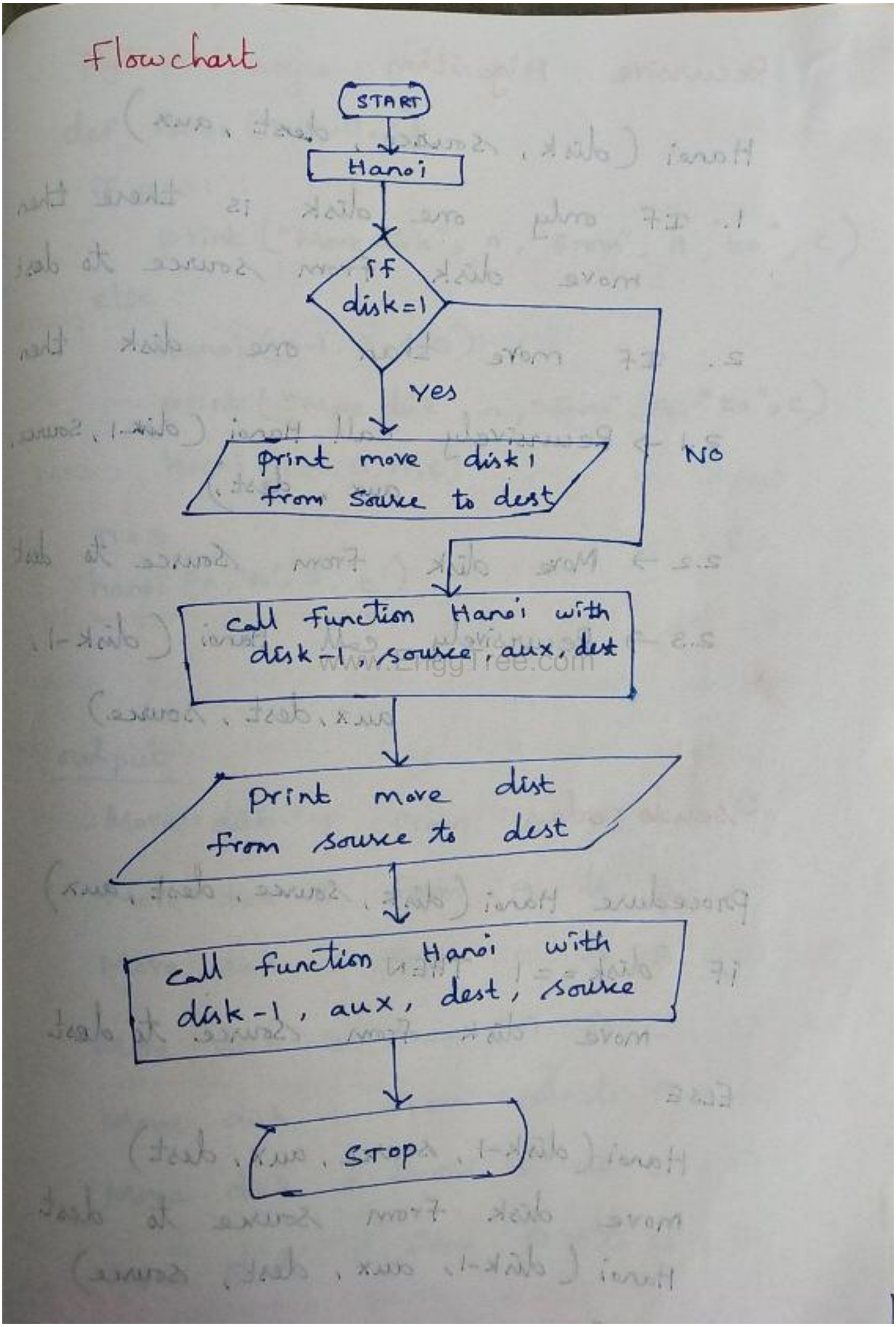
The Tower of Hanoi problem is to find a sequence of disk moves so that all the disks moved from stand 1 to stand 3 adhering following rules

1. Move only one disk at a time
2. A larger disk cannot be placed on top at a smaller disk.
3. All disks except the one being moved should be on a stand.





Flowchart



Recursive Algorithm

Hanoi (disk, source, dest, aux)

1. If only one disk is there then move disk from source to dest
2. If more than one disk then
   - 2.1 → Recursively call Hanoi (disk-1, source, aux, dest)
   - 2.2 → Move disk from source to dest
   - 2.3 → Recursively call Hanoi (disk-1, aux, dest, source)

Pseudo code

```
Procedure Hanoi (disk, source, dest, aux)
IF disk == 1 THEN
    move disk from source to dest
ELSE
    Hanoi (disk-1, source, aux, dest)
    move disk from source to dest
    Hanoi (disk-1, aux, dest, source)
END IF
END procedure.
```

Python program

```
def hanoi (n, A, B, C)
  if n==1 :
     print ("Move disk", n, "from", A, "to", C)
  else :
     hanoi (n-1, A, C, B)
     print ("Move disk", n, "from", A, "to", C)
     hanoi (n-1, B, A, C)
```

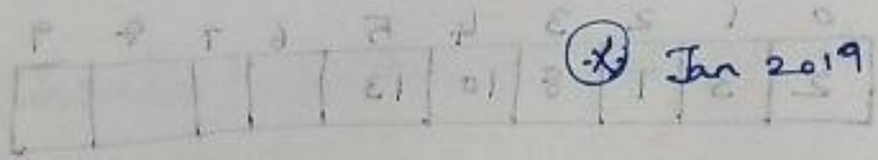
n=3

```
hanoi (n, 'A', 'B', 'C')
```


output

```
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
```

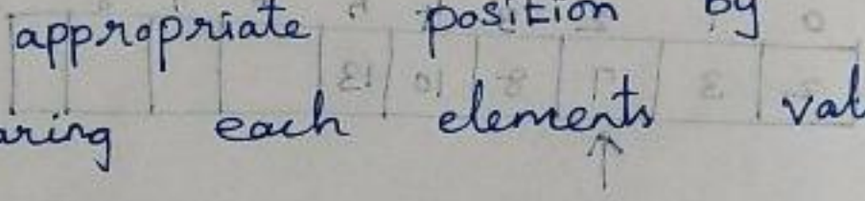
Insert a card in a list of sorted cards



(X) Jan 2019 (8 Mark)

Description

1. To insert a card in the sorted card, we must increase the list size with 1
2. We can insert a new card in the appropriate position by comparing each elements value.



Case(i)

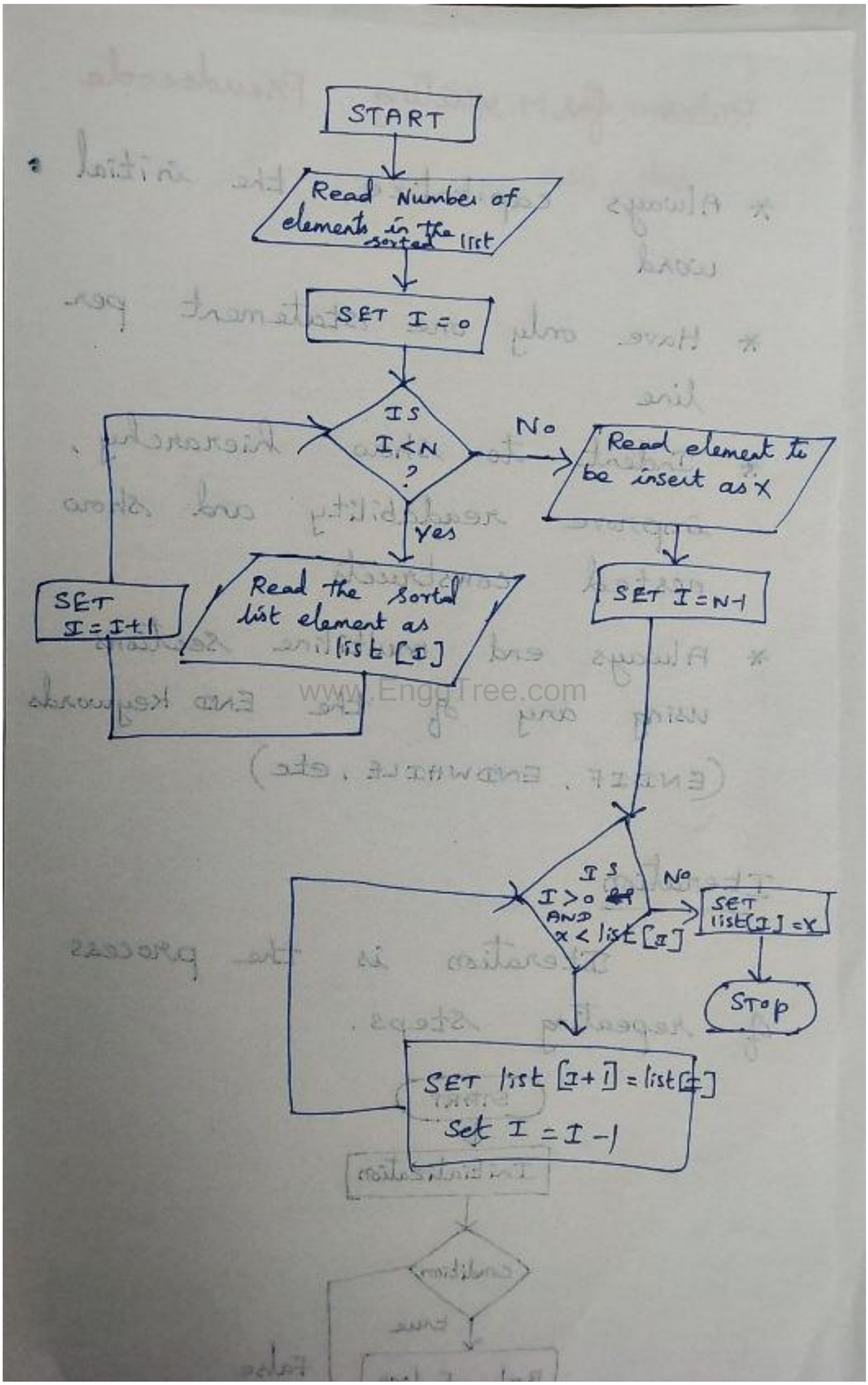
If newcard < element value

Move the element by one position up

Case(ii)

If newcard > element value

place this newcard in the current value of I+1



Diagramatic Representation N=5

List Index	List[0]	List[1]	[2]	[3]	[4]	[5]	Element to be inserted
Original cards	3	5	7	9	11		10

I = n-1 = 4

I > 0 && x < list[I]

4 > 0 10 < list[4]

10 < 11

original cards 3 5 7 9 11

↑

I

3 5 7 9 10 11

x > 0 && x < list[I]

3 > 0 &&

10 < 9.

Pseudo code:

READ number of elements in the sorted list as N

Set I=0

WHILE I<N

Read the sorted list element as list[I]

Set I=I+1

READ element to be inserted as X

SET I = N-1

WHILE I >= 0 AND X < LIST[I]

LIST [I+1] = LIST[I]

SET I = I-1

LIST [I+1] = X

END

Algorithm

Step 1 : Start

Step 2 : Read the no of elements in the sorted list as N

Step 3 : Initialize I = 0

Step 4 : Repeat steps 5 and 6 WHILE I < N

Step 5 : Read the sorted list element as list [I]

Step 6 : set I = I+1

Step 7 : Read the element to be inserted as X

Step 8 : Set I = N-1

Step 9 : Repeat step 9, 10, 11 WHILE I >= 10 and X < list [I]

Step 10 : List [I+1] = List [I]

Step 11 : Set I = I - 1

Step 12 : ~~Stop~~ List [I+1] = X

Step 13 : Stop

Guess an integer number in a range

problem statement:

The objective is to randomly generate integer number from 0 to n. Then the player has to guess the number correctly, output an appropriate message.

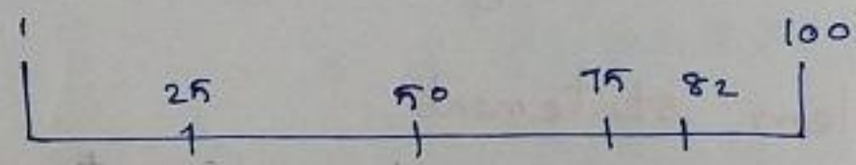
If the guessed number is less than the random number generated, output the message "Your guess is lower than the number, guess again".

Otherwise output the message "Your guess is higher than the number guess again"

Then the player guess another number. This process is repeated until the player guess the correct number.

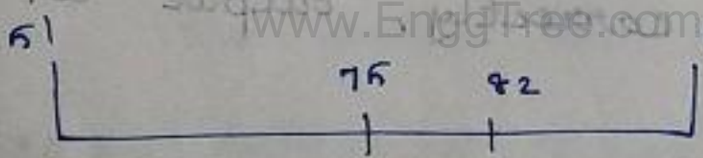
Example

Game - Secret number is 82



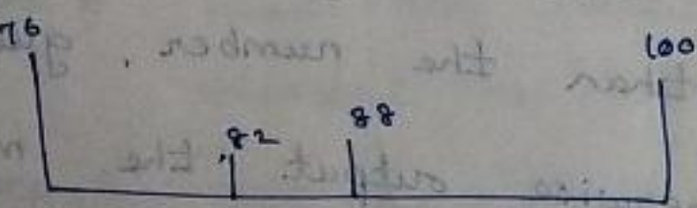
First guess = 50 → Middle value

Answer = Too low



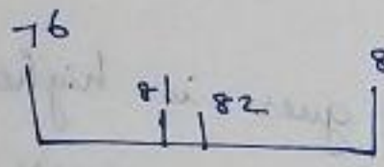
Second guess = 75

Answer = low guess again



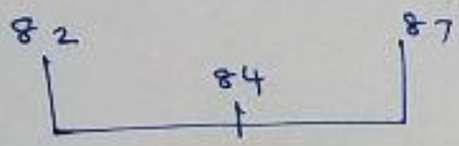
Third guess 88

Answer : High guess again



Fourth guess : 81

Answer : low guess again



Fifth guess : 84

Answer : High guess again

Sixth guess : 82

Answer : correct

Algorithm

Step 1. Start

Step 2. Generate a random number and read num

a. Enter the number to guess

b. if (Guess is equal to num)

print (" You guessed the correct number")

otherwise

if (Guess is less than num)

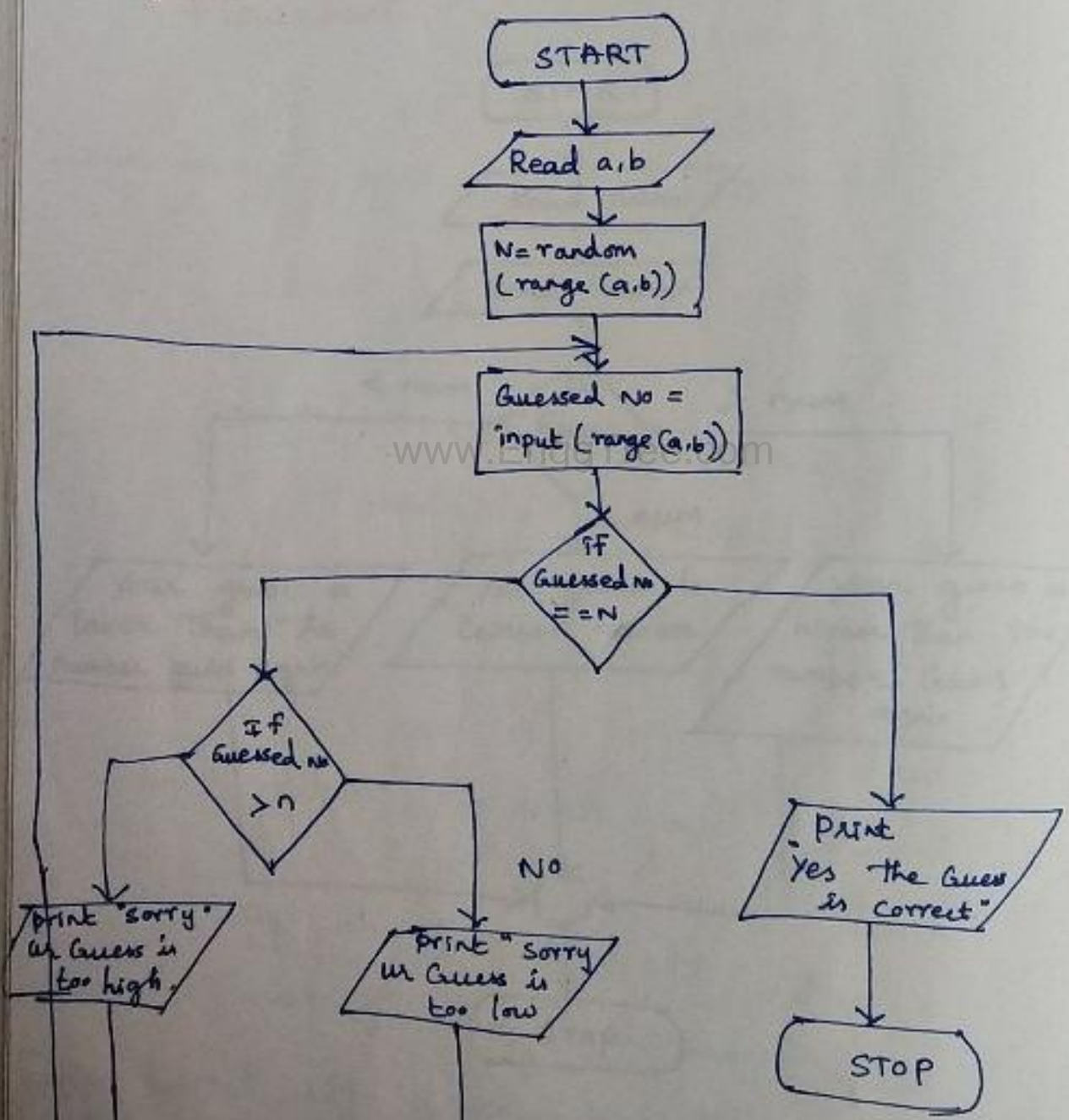
print (" Your guess is lower than the number . guess again")

otherwise

print ("Your guess is higher than the number, guess again")

step 3 : Stop

Flowchart



Pseudocode

```
BEGIN
COMPUTE N = random value in range
READ guess
IF guess = N then
print ("Your guess is correct")
ELSE
print ("Your guess is incorrect"
        Try again")
END
```

Find minimum element in list of items

A be an array containing a list of items. Let us assume that the first item is the smallest item (~~a[i]~~ min = a[0])

* Then compare all the remaining items with the smallest item (a[i] < min)
* If the new item is smaller than our assumption, update min with this new item as the smallest item in the list (min = a[i])
* Continue till all items in the list are checked.

Algorithm

Step 1 : Start

Step 2 : Create list a = [3,2,1]

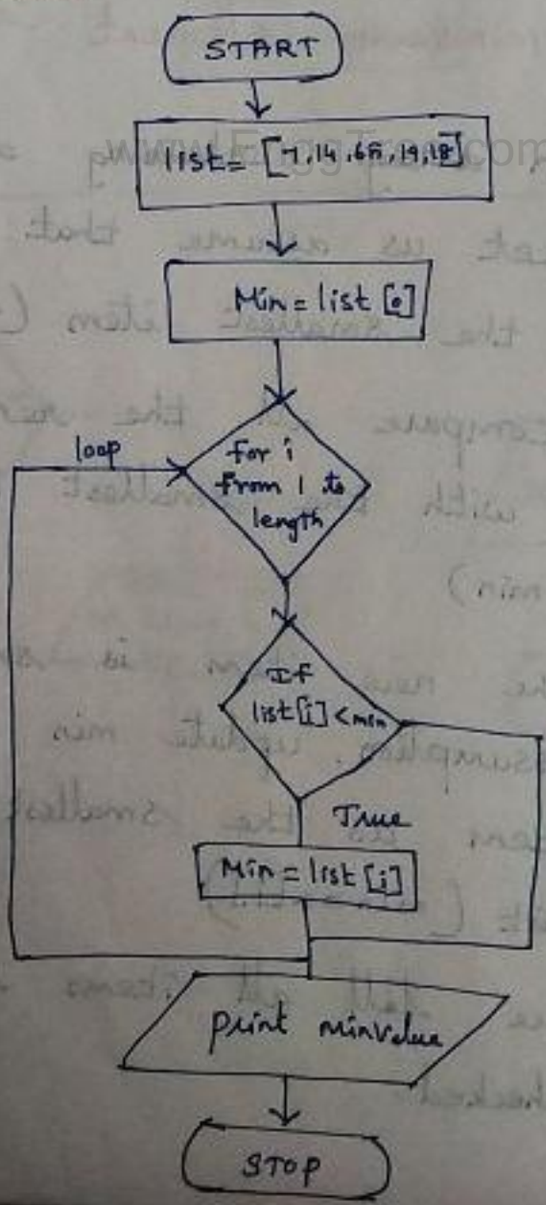
Step 3 : minvalue = min (a) find the minimum value of a list

Step 4 : minindex = a.index (minvalue)
find the index of the minimum value.

Step 5 : print minvalue and minindex

Step 6 : Stop

Flowchart



Pseudocode

```
BEGIN
    CREATE list
    Find the minimum value using
        b = min(a)
    PRINT b
END
```


UNIT - II

Data types, Expressions, Statements

python Interpreter and Interactive Mode

python is an easy to learn, powerful programming language.

* It has efficient high level data structures and a simple but effective approach to object oriented programming

Interactive Mode of Programming

* python provides interactive computing facility.

* Type the command at the python prompt and execute the commands directly for most of the instructions

Few instruction in the IDE

```
>>> 5+10
15
>>> print("My python Program")
My python Program
```

* The command is the prompt on execution produces the output immediately
* Since python interprets instruction line by line and produces output.

Script Mode of Programming

Invoking interpreter with a script begins execution of the script and continues until the end of the script.

* When the script is finished the interpreter is no longer active
* The python scripts should be saved with the extension aa.py

Syntax and Semantics

Syntax and semantics are important concepts that apply to all languages.

Syntax

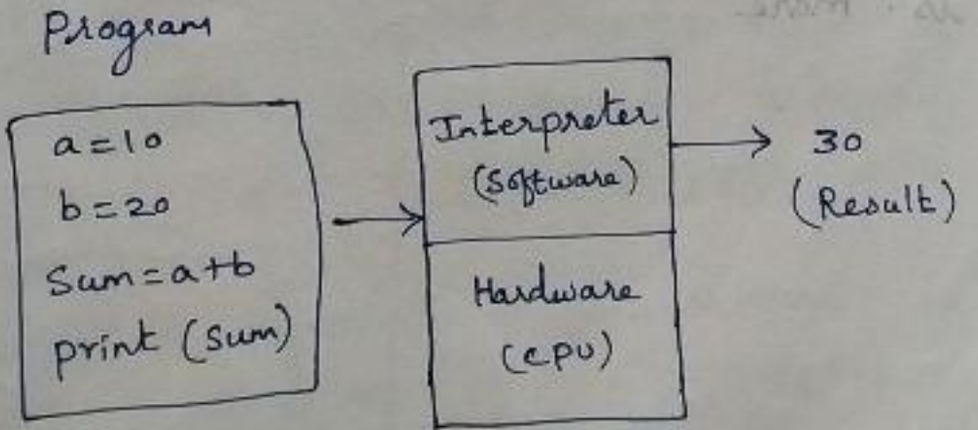
The syntax of a language is a set of characters and the acceptable sequence of those characters.

Semantics

The semantics of a language is the meaning associated with each syntactically correct sequence of characters.

Interpreter

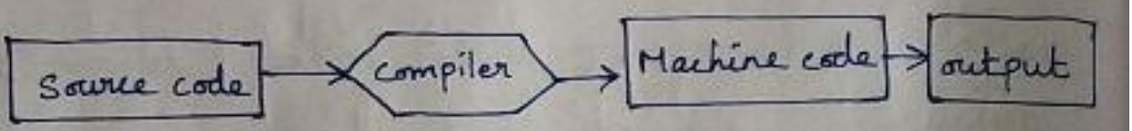
Interpreter is one type of translator, which executes program instructions line by line in place of the CPU.



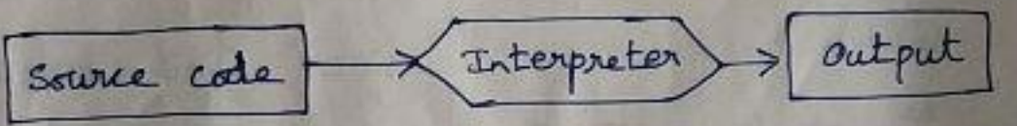
Compiler Vs Interpreter

8 Mark
Jan 2019

How compiler Works



How Interpreter Works



Compiler	Interpreter
1. The Compiler takes a program as a whole and translates it.	Interpreter translates a program statement by statement.
2. Intermediate code or machine code is generated in case of a compiler.	Interpreter does not create intermediate code.
3. A compiler is comparatively faster than interpreter as the compiler take the whole program.	Interpreters compile each line of code after the other. So It is slow.

	Compiler	Interpreter
5.	In Compiler when an error occurs in the program, it stops its translation and after removing error whole program is translated again	When an error takes place in the interpreter, it prevents its translation and after removing the error, translation resumes
6.	In a compiler the process requires two steps in which firstly source code is translated to target program then executed.	Interpreter is a one step process in which source code is compiled and executed at the same time.
6.	The compiler is used in programming languages like C, C++ etc..	Interpreter is used in languages like PHP, python etc..

Python has two basic modes

1. Interpreter mode
2. Interactive mode

1. Interpreter mode

The interpreter mode is the mode where the scripted and finished .py files are run in the python interpreter.

2. Interactive mode

In interactive mode is a command line shell which gives immediate feedback for each statement fed in the active memory.

* As new lines are fed into the interpreter,
* The fed program is evaluated both in part and in whole.

Python Interactive Mode

```
Python 3.6.0 (default Dec 2, 2021 : 9:16 :22)
on Windows 7
```

* The first three lines contain information about the interpreter and the operating system it is running on.

* The version number is 3.6.0 It begins with 3, if the version is Python 3.
* The last line >>> is a Prompt that indicates that the interpreter is ready to enter code.
* If the user types a line of code and hits Enter, the interpreter displays the result.

Example

```
>>> 5+7
12
>>> print("Grace")
Grace
>>> num=10
>>> num=num/2
>>> print(num)
5.0
```

Python Interpreter mode

* The python interpreter is a program that reads and executes python code, in which python statements can be stored in a file.
* The python system reads and executes the commands from the file, rather than from the console.
* Such a file is termed as python program

```
Microsoft Windows 7
c:\Users\admin\python
Python 3.9.5
>>> a=2
>>> b=3
>>> c=a+b
>>> print(c)
5
```

* The first three lines contain information about the interpreter and the operating system it is running on.

* The version number is 3.6.0. It begin with 3, if the version is python 3.

* The last line >>> is a prompt that indicates that the interpreter is ready to enter code.

* If the user type a line of code and hits Enter, Interpreter displays the result.

Types

Data type of an object determines what values it can have and what operations can be performed on it.

Data type is a category of values

The built in datatypes are

* Numbers (integer type, Floating point type, and complex type)
* String
* List
* Tuple
* Set
* Boolean
* Dictionaries

Numbers

Number datatypes store numeric values.

* python has three number types
  * Integer numbers
  * Floating point numbers
  * Complex numbers

a) Integer Type

An integer type (int) represents signed whole numbers An integer represents both positive and negative numbers.

To write an int constant

* A zero is written as just 0
* To write an integer in decimal (base 10) the first digit must not be zero.
  eg) 25000
* To write an integer in octal (base 8), precede it with "0" (zero) and use the digits 0 to 7 plus A, B, C D, E, F
  Eg. 0_0145
* To write an integer in hexadecimal (base 16) precede it with "0x" (zero)
  Eg : 0x77

* To write an integer in binary (base 2) precede it with "0b" or "0B"

Example : 0b101

Floating Point Type

A floating point (float) type represents numbers with fractional part.

* A floating point number has a decimal point and fractional part.

Example : 3.0 or 3.7 or -28.72

python cannot represent very large or very small numbers and the precision is limited to only about 14 digits

* A floating point also represents scientific notation. It is stored with three parts
  * A sign + or -
  * A mantissa
  * An Exponent

eg: 1.6E3 stands for 1.6×10^3

Complex Type

A complex data type is an immutable type that holds a pair of numbers, one representing the real part and the other representing the imaginary part of a complex number.

* Complex number are written with the real and imaginary parts joined by a + or - sign, with the imaginary part followed by a j

Example

5+14j

python displays complex numbers in parentheses when they have a nonzero real part.

Example

```
>>> z=5+14j
>>> z.real
5.0
>>> z.imag
14.0
```

```
>>> print (z)
(5+14j)
>>> print ((2+3j) * (4+5j))
-7+22j
```

type() function

Used to check the type of any value / variable.

```
>>> type (5)
<class 'int'>

>>> type (3.14)
<class 'float'>

>>> type ('hello')
<class 'str'>
```


Variables

Variables are reserved memory location to store values

* A variable is a name that refers to a value
* An assignment statement creates new variables and gives them values.

Variable

```
>>> msg = "Hello"
>>> n=17
>>> a = 3.14
>>> b, c = 4 , 1.2
>>> type (msg)
 <class 'str'>
>>> type (a)
 <class 'float'>
```

Data Types / Types

* Every value in python has a data type
* Since everything in an object is python.
  → Datatype - classes
  → Variables — instances (objects) of these classes
* But in python no need to declare data type of the variables.
* Data type determines how the value can be used.

Data Types in Python

1. Numbers
   - i. int
   - ii. long
   - iii. Float
   - iv. Complex
2. String
3. Boolean

(1, 2, 3 — Basic Data Types)

4. List
5. Tuple
6. Dictionary
7. Set
8. Byte & Byte array
9. None

Numbers

* Created with numeric literals
* Numeric object are immutable.
  ie) when an object is created its value cannot be changed.

```
>>> a = 4
>>> id(a)
164992080
>>> a = 5
>>> id(a)
164992288
>>> a = 4
>>> id(a)
164992080
```

Numberic types in python

* int
* long
* float
* complex.

int

* Integers are whole numbers without decimal point (+ve/-ve)
* Integers have unlimited size in python (no limits)

Ex: a = 9293 (+ve Decimal number)
b = -38 (-ve Decimal number)
c = 0x9A (Hexadecimal number)
d = 0056 (Octal number)
e = 0b1111 (Binary number)

Str

Strings are immutable

ie) characters can be accessed but cannot be modified

```
>>> s = "hello"
>>> print (s[0])
h
>>> s[0] = "H"
Type Error : 'str' object does not support item assignment
```

→ we can check the index value but we can't change it → immutable

```
>>> s = "Hello"
>>> id (s)
66374816
>>> s = "hello"
>>> id (s)
66220576
```

bool

Boolean is a datatype having two values True and False

* It is the result of conditional statements.

Ex

```
>>> 2==2
True
>>> a=(2>3)
>>> print(a)
False
```


List

Ordered sequence of values separated by commas and enclosed within square brackets []

* List are mutable

The values belonging to a list can be of different datatype

```
>>> a=[]
>>> b=[1,3,25,7]
>>> c=[45,"John"]
```

Mutable

```
>>> l = [1, 2, 3.5, "11"]
>>> id(l)
66831460
>>> print(l)
[1, 2, 3.5, '11']
>>> l.append(7)
>>> print(l)
[1, 2, 3.5, '11', 7]
>>> id(l)
66831460
```

Same id.

Tuple

* ordered sequence of values separated by commas and enclosed within parentheses ()
* Main difference between list and tuple is : Tuple is immutable
  ie) Cannot be updated
  Elements and size can't be changed.
  → Can be through as a read only list.

Ex: a=(5, 6.5, 0)

b=(45, "Grace")

Dict (dictionary / Mapping)

* Dictionary are lists of key : value pairs
* Used to store a lot of related information that can be associated through keys.
* It is used to extract a value based on the key name.
* Created using braces {} with pairs separated by a comma (,) and the key values associated with a colon (:)
* Dictionaries are mutable
* unlike lists, where index numbers are used dictionary allow the use of key to access its members
* key must be unique

Ex:
```
>>> room = {'John': 25, 'Tom': 35}
>>> room['John']
25
```

d = {12 : 'Tom', 15: 'Jerry', 10 : 'Grace'}
(12 ↓ Key, 'Tom' ↓ value)

```
>>> type (d)
<class 'dict'>
>>> d [12]
'Tom'
```

We can't access by index, only access by key

```
>>> d [0]
```

Error will raise

Set

* unordered collection of unique and immutable objects

```
>>> s = {1, 6, 9, 3}
>>> type (s)
<class 'set'>
>>> s
{1, 3, 6, 9}
>>> s = {'a', 'g', 'e', 'a'}
```

```
>>> s
{'a', 'e', 'g'}
```

If the value is repeated it will remove automatically.

→ output is unordered.

None

None is a special constant in python

* Null Value (not 0, not empty string, not false)

Eg

```
>>> x = None
>>> type(x)
<class 'NoneType'>
>>> type(None)
<class 'NoneType'>
```

Variables

* Variables are containers for storing data values.

Creating Variables

A variable is created the moment you first assign a value to it.

Example

```
x = 5
y = "John"
print(x)
print(y)
```

output

```
5
John
```

Variables do not need to be declared with any particular type.

* Even change type after they have been set.

Example

```
x = 4          # x is of type int
x = "Grace"    # x is now of type str
print(x)
```

output

Grace

Casting

If you want to specify the data type of a variable, this can be done with casting.

Example

```
x = str(3)
y = int(3)
z = float(3)
print(x)
print(y)
print(z)
```

output

```
3
3
3.0
```

Get the Type

You can get the datatype of a variable with the type() function

Example

```
x=5
y="John"
print(type(x))
print(type(y))
```

output

```
<class 'int'>
<class 'str'>
```


Single or Double Quotes

String variables can be declared either by using single or double quotes

Example

```
x = "Grace"
# is the same as
x = 'Grace"
```

output

```
Grace
Grace
```

Case Sensitive

Variable names are case-sensitive

Example

```
a = 4
A = "Grace"
# A will not overwrite a
print (a)
print (A)
```

output

```
4
Grace
```


Rules for python variables

* A variable name must start with a letter or the underscore character
* A variable name cannot start with a number.
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9 and _)
* Variable names are case sensitive (age, Age and AGE are three different variables)

Example

```
myVar = "Grace"
my_var = "Grace"
_my_var = "Grace"
myVar = "Grace"
MYVAR = "Grace"
myvar = "Grace"
```

output

```
Grace
Grace
Grace
Grace
Grace
Grace
```


Note : Variable names are case sensitive

Multi Words Variable Names

Camel Case

Each word, except the first starts with a capital letter:

```
myVariable = "Grace"
```

Pascal Case

Each word starts with a capital letter

```
MyVariable = "Grace"
```


Snake Case

Each word is seperated by an underscore character

```
my_variable = "Grace"
```

Python Variables - Assign Multiple Variables

Python allows you to assign values to multiple variables in one line:

Example

```
x, y, z = "Orange", "Mango", "Apple"
print(x)
print(y)
print(z)
```

output

```
orange
Mango
Apple.
```

Note: Make sure the number of variables matches the number of values or else you will get an error.

One Value to Multiple Variables

You can assign the same value to multiple variables in one line

Example

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

o/p

```
Orange
Orange
Orange
```

unpack a collection

If you have a collection of values in a list, tuple etc,

* python allows you to extract the values into variables.
* This is called unpacking

Example

```
Fruits = ["apple", "banana", "cherry"]

x, y, z = Fruits
print (x)
print (y)
print (z)
```

output

```
apple
banana
cherry
```

Output Variables

The python print statement is often used to output variables.

* To combine both text and a variable, Python uses the + character

Example

```
x = "Grace"
print ("Python is " + x)
```


output

Python is Grace

You can also use the + character to add a variable to another variable

Example

```
x = "Grace"
y = "College"
z = x + y
print (z)
```

output

Grace College

For number the + character works as a mathematical operator

Example

```
x = 5
y = 10
print (x+y)
```

output

15

IF you try to combine a string and a number, python will give you an error

```
x = 5
y = "Grace"
print (x + y)
```

output

TypeError: unsupported operand type(s) for +: 'int'

Python - Global Variables

Variables that are created outside of a function are known as global variables

* Global variables can be used by everyone both inside of functions and outside.

Example

Create a variable outside of a function and use it inside the function.

```
x = "awesome"  -> Global Variable
def myfunc():
    print ("python is " + x)
myfunc()
```

output

python is awesome.

Example

Create a variable inside a function with a same name as the global variable.

```
x = "awesome"  → Global Variable
def myfunc():
    x = "fantastic"  → local variable
    print("python is " + x)

myfunc()
print("python is " + x)
```

output

```
python is fantastic
python is awesome
```

local variable is used within the function

The Global Keyword

* When you create a variable inside a function, that variable is local and can only be used inside the function.

* To create a global variable inside a function, you can use the global keyword.

Example


```
def myfunc():
    global x
    x = "fantastic"

myfunc()
print("python is " + x)
```

Output

```
python is fantastic
```

Python Variables

Python Variables also use the global keyword if you want to change a global variable inside a function.

Example

```
X = "awesome"
def myfun():
    global x
    x = "fantastic"
myfun()
print("python is " + x)
```

output

```
python is fantastic
```

Always inside the function will be executed

Expression and Statements

Expression

An expression is a combination of values, variables and operators

* A value all by itself is considered an expression and so is a variable.

Legal Expressions


```
>>> 42
42
>>> n
17
>>> n+25
42
```

When you type an expression at the prompt, the interpreter evaluates it.

Statements:

A statement is a limit of code that has an effect like creating a variable or displaying a value.

```
>>> n=17
>>> print(n)
```

→ The first line is an assignment statement that gives a value to n

→ The second line is a print statement that displays the value of n.

→ When you type a statement, the interpreter executes it, which means that it does whatever the statement says

→ In general, statements don't have values.

Tuple Assignment (X)

Tuple

The tuple is used to store sequence of items

* The tuple are separated by commas and Not necessary enclosed within parentheses ( )
* Tuple is the name of a built in function.

Tuple Assignment with An Example:

1. To swap a and b:

The conventional assignment to use a temporary variable.

```
>>> temp = a
>>> a = b
>>> b = temp
```

This solution is Cumbersome

Tuple assignment is more elegant

```
>>> a, b = b, a
```

a, b → Variable
b, a → Tuple of expression

* The left side is a tuple of variables
* The Right side is a tuple of expression.
* Each value is assigned to its respective variable.
* All the expression on the right side are evaluated before any of the assignment.

Example

The number of variables on the left and the no of values on the right have to be the same.

if it is more error.

```
>>> a, b = 1, 2, 3
```

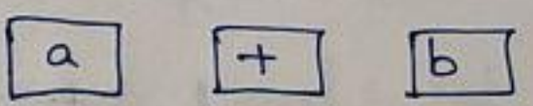
ValueError: too many values to unpack.

Precedence of operators (-X-)

16 Mark

OPERATORS

Ex: a + b



first operand — operator — second operand

Types of operators

1. Arithmetic operators
2. Comparison (Relational operator)
3. Logical operators
4. Assignment operators
5. Bitwise operators
6. Membership operators
7. Identity operators

1. Arithmetic Operators

Y = 20
X = 10

operator	Description	Example
+	Add	X + Y = 30
-	Sub	X - Y = -10
*	Mul	X * Y = 200
/	Division	Y / X = 2
%	Modulus	Y % X = 0
**	Exponent (power)	X ** Y x^y
//	Floor Division (Removes decimal point)	9 // 2 = 4

2. Relational Operators

Operator	Description	Example
==	2 operands equal condition True	x==Y is True
!=	If 2 operands are not equal condition True	x!=Y True
>	Greater	X>Y =T
<	Lesser	X<Y =f
>=		X>=Y =T
<=		X<= Y =f

3. Logical Operators

Operator	Description	Example
and Logical AND	If both operands are true Condition is True	(x and y) is True
or logical OR	If any of the 2 are True Condition True	(x or y) is True
not Logical NOT	To reverse the logical state of its operand	Not (x and y) is False

4. Assignment operator

operator	Description	Example
=	Assign values from Right operands to left side	z = x+y assign value of x+y into z
+= Add	It adds right operand to the left & assign result to left operand	y+=x is equivalent to y=y+x

operator	Description	Example
- = Sub	It subtracts right operand from the left operand and assign the result to left operand.	y -= x is equivalent to y = y - x
* = Multiply		y *= x y = y * x
/ = Divide		y /= x y = y / x
% = Modulus		y %= x y = y % x
** = Exponent		y **= x is equivalent to y = y ** x
// = Floor division		y //= x is equivalent to y = y // x

Bitwise Operators

operation at Bit level, treats int as bits

operator	Description	Example
&	Bitwise AND	1101 & 1111 = 1101
|	Bitwise OR	1101 & 1111 = 1111
^	Bitwise XOR	1101 & 1111 = 0000
~	Bitwise NOT	Reduces 1's complement of the resultant

	8	4	2	1
4 =	0	1	0	0

Membership operators

operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence & false otherwise.	x in y, here in results a 1 if x is a member of sequence y
Not in	Returns True, if a sequence with the specified value is not present in the object	x not in y

Identity operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object with the same memory location.

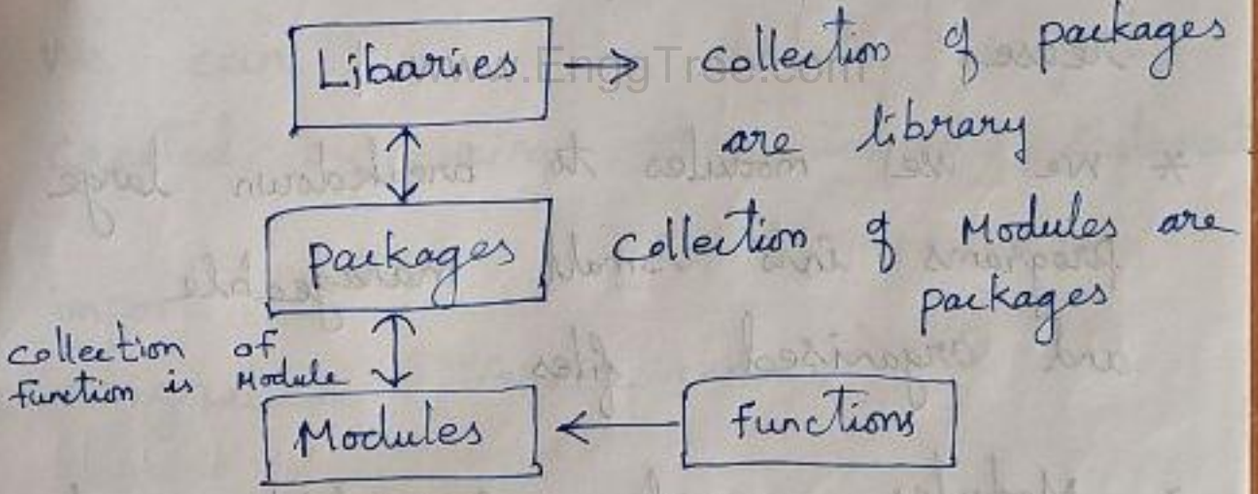
Operator	Description	Example
is	Returns true if both variables are the same object	x is y
is not	Returns true if both variables are not the same object	x is not y

Module

Modules are previously written piece of code that are used to perform all common tasks required frequently instead of creating the functions again and again.

Example : All mathematical operations are saved in math module.

Diagram to understand the concept of function Modules and Packages



Modules breakdown complex programs into simple manageable files / functions

Modules and Functions

* Module is a file containing python definitions and statements.
* A file containing python code, for eg. example.py is called a module and its module name would be example
* Function helps to reuse a particular piece of code, Module reuse
* We use modules to breakdown large programs into small manageable and organised files
* Modules provide reusability of code
* We can define our most used functions in a module and import it instead of copying their definitions and code in different programs.

Creating a Module [User defined]

To create a module just save the code you want in a file with the file extension .py

aa.py

```
def greeting(name):
    print("Hello," + name)
```

Use a Module

We can use the module we just created by using the import statement

```
import aa
aa.greeting("Grace college")
```

output

```
Hello, Grace College
```

Variables in Module

The module can contain functions but also variables of all types (arrays, dictionaries, objects etc)

Example

aa.py

```
person = {"name": "Jeffrin", "age": 7,
          "country": "India"
        }
```


import the module named aa and access the person dictionary

```
import aa
a = aa.person["age"]
print(a)
```

Output

7

Re-naming a Module

You can create an alias when you import a module by using the as keyword.

Example

Create an alias for aa as bb

```
import aa as bb
a = bb.person["age"]
print(a)
```


Output

7

Built-in Modules

There are several built in Modules in python

Example

```
import platform
x = platform.system()
print(x)
```

Output

Windows

Function Definition

Functions are group of related statements that perform a specific task.

* Functions help us to break our program into smaller and modular chunks. as our program grows larger and larger functions make it more organized and manageable.
* Function avoids repetition and makes code reusablity.

Syntax of functions

```
def function name (parameters):
    """docstring"""
    statements
```

1) Keyword def marks the start of function header.
2. A function name to uniquely identify it

3. parameters (arguments) through which we pass values to a function.

4. : to make the end of function header

5. optional return statement to return a value from the function.

6. optional documentation string to describe what the function does

7. One or more valid python statements that makeup the function body.

8. Statement must have same indentation level.

Function Types

1. Built-in-function

   Example : python packages & Libraries

2. User defined function

   Example : Developer write to meet certain requirement

Python Built in Functions

* The python interpreter has a no of functions that are always available for use.
* These functions are called Built in functions
* Python 3.6 has 68 Built in functions

Sample Built in functions are

Python abs() : returns absolute value of a no

python bool() : converts a value to Boolean.

python input() : reads and return a line of string

python list() : Create list in python

python max() : returns largest element
min() : returns smallest element
slice() : creates a slice Object specified by range()

User - Defined Function

* The function that readily come with python are called built - in - functions, if we use functions written by others in the form of library it can be termed as library functions

* All the other functions that we write on our own fall under user defined function.

Example

→ Parameter / formal values

```
def add_number(x,y):
    Sum = x+y
    return Sum

num1 = 5
num2 = 6
print("The sum is", add_number(num1,num2))
```

Arguments / Actual values

Advantages of User Defined Function

* Easy to understand, maintain and debug.
* Programmers working on large project can divide the workload by making different functions
* If repeated code occurs in program Functions can be used to include those codes & execute when needed by calling the functions.

Success
DATE:
PAGE:

Illustrative programs

Exchange the Values of two variables

Swap Two variables using a temporary (third) Variable

```
temp = a
a = b
b = temp
```

Python Program


```
a = int (input ("Enter first number : "))
b = int (input ("Enter second number : "))
print ("Before swapping a= ", a, "and b = ", b)
temp = a
a = b
b = temp
print (" After swapping a= ", a, "and b= ", b)
```

Output

```
Enter first number : 5
Enter second number : 6
Before swapping a=5 and b=6
After swapping a=6 and b=5
```

only used in python

Swap two variables without using a temporary variable.

```
a = int (input ("Enter first number : "))
b = int (input ("Enter second number : "))
print ("Before swapping a = ", a, " and b = ", b)

a , b = b, a

print ("After swapping a = ", a, " and b = ", b)
```

Output

```
Enter First number : 4
Enter Second number : 1
Before swapping a = 4 and b = 1
After swapping a = 1 and b = 4
```

Using functions

```
def swap(a,b):
    print ("Before swapping a= ", a, " and b= ", b)
    a, b = b, a
    print ("After swapping : a= ", a, " and b= ", b)

a = int (input ("Enter first number : "))
b = int (input ("Enter second number : "))
swap(a, b)  --> call the function
```

Output

```
Enter first number : 4
Enter Second number : 5
Before swapping a = 4 and b = 5
After swapping a = 5 and b = 4
```

Circulate the values of n variables

Program

```
a = list(input("Enter the list "))
print(a)
for i in range(1, len(a), 1):
    print(a[i:] + a[:i])
```

Output

```
Enter the list 1234
['1', 2, 3, 4]
[2, 3, 4, 1]
[3, 4, 1, 2]
[4, 1, 2, 3]
```


List program to circulate the value of n variables

```
def rotate(l,n):                          # l, i
    newlist = l[n:] + l[:n]
    return newlist
list = [1,2,3,4,5]                        # index 0 1 2 3 4
print("Original list = ", list)
mylist = rotate(list,1)
print("List rotated by 1 = ", mylist)
mylist = rotate(list,4)
print("List rotated by 4 = ", mylist)
```

→ Parameter / Formal parameter

```
l[1:] + l[:1]
[2,3,4,5] + 1
newlist = 2,3,4,5,1
```

→ Arguments / Actual parameter

```
l[4:] + l[:4]
5 + 1,2,3,4
newlist = 5,1,2,3,4
```

output

```
Original list = [1,2,3,4,5]

List rotated by 1 = [2,3,4,5,1]
List rotated by 4 = [5,1,2,3,4]
```

Success
DATE:
PAGE:

Distance between Points

The distance formula is derived from the Pythagarean theorem.

To find the distance between 2 points (x_1, y_1) and (x_2, y_2) then the following formula is used.

$$\text{Distance} = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$$

Program

```
import math
P1 = [2,4]          # x1, y1
P2 = [3,6]          # x2, y2
distance = math.sqrt (((P2[0] - P1[0]) **2) +
                      ((P2[1] - P1[1]) **2)))
print ("The distance between 2 points", distance)
```

Output

The distance between 2 points 2.2360679775

Success
DATE:
PAGE:

UNIT-III

Control flow, functions, Strings

1. Conditionals: Boolean values and operators
2. Conditional (if), alternative (if-else), chained conditional (if..... elif..... else)
3. Iteration:
   i) State
   ii) While
   iii) For
   iv) break, continue, pass
4. Fruitful functions:
   i) return value
   ii) Parameters
   iii) local and global scope
   iv) function composition
   v) Recursion.
5. String:
   i) String slice
   ii) Immutability
   iii) String functions & methods
   iv) String Module
6. List: As an array
7. Illustrative Programs
   1. Square root
   2. gcd
   3. Exponentiation
   4. Sum of an array of numbers
   5. Linear Search
   6. Binary Search

Conditionals : Boolean Values And operators

Boolean Values And Operators

* The Boolean Data type is a data type, having 2 values usually denoted as True and Fal
* When converting a bool to an int, the integer value is always 0 or 1.
* But when converting an int to bool, the boolean value is True for all integer except 0

Boolean And Logical Operators

Boolean values respond to logical operators 'and' 'or'

True and False = False
True and True = True
False and True = False
False or True = True
False or False = False

T F
1 * 0 = 0
1 + 0 = 1
and → *
or → +

T F F
1 * 0 = 0
T T T
1 * 1 = 1
F T F
0 * 1 = 0

F T T
0 + 1 = 1
F F F
0 + 0 = 0

Conditional Statements

The statements used for decision making are called as 'Conditional Statements', They are also known as Conditional expressions or Conditional constructs.

Types of Conditional Checking Statements

Python programming language provides the following types of conditional checking statements.

1. if statements
2. if .... else statements
3. if .... elif .... else statement
   or
   chained condition

Rules

* The colon(:) is required at the end of the condition.
* The body of the if statement is indicated by the indendation
* In python 4 spaces are used for indendation
* All lines indended the same amount after colon will be executed for true condition.
* Python interprets non-zero values as True. None & 0 are interpreted as False.

PAGE

Following this Agenda to write if, if...else & if...elif...else

* Statement Explanation
* Syntax
* Flowchart / State Diagram.
* Example program.

If Statement

Statement Explanation

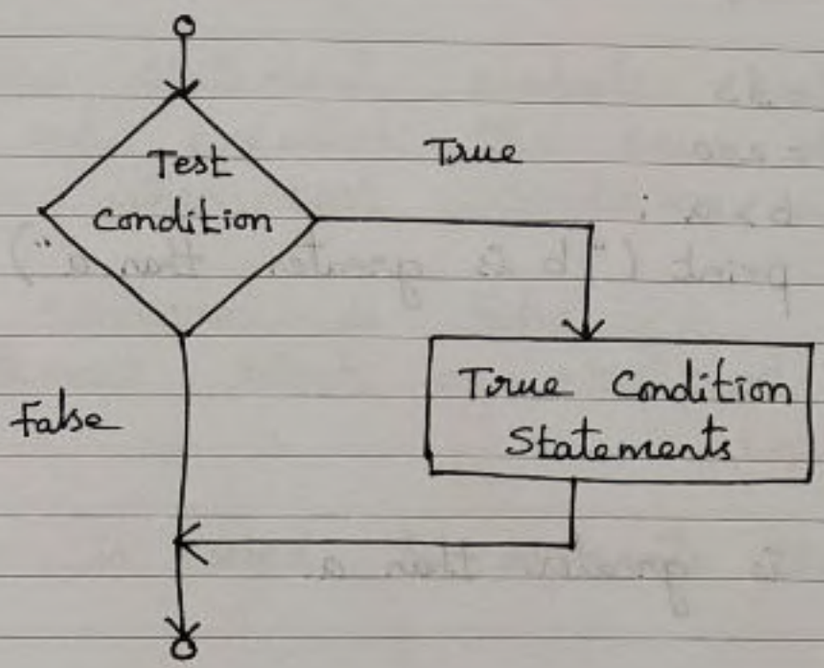
* In "if" statement, the test expression is evaluate first, if the test expression is "True", the statements which are indented and below 'if' structure are executed.

* if the test expression is "False", the Control will flow to the next statement after the 'if' statement

Syntax

```
if test_expression :
    Statement (s)
```

Flowchart



Example program

```
num = 3
if num > 0 :
    print (num, " is a positive no")
print ("This is always printed")

num = -1
if num > 0 :
    print (num, " is a positive no")
print ("This is also always printed")
```

Output

```
3 is a positive no
This is always printed
This is also always printed
```

PAGE:

Example

```
a = 33
b = 200
if b > a :
   print ("b is greater than a")
```

Output

b is greater than a.

Indentation

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code.

Example
if statement without indentation
(will raise an error)

```
a = 33
b = 200
if b > a :
print ("b is greater than a")
```

Output

indentation error : expected an indented block.

PAGE:

if....else Statements:

The if...else statement evaluates test condition and executes the true statement block only when test condition is True.

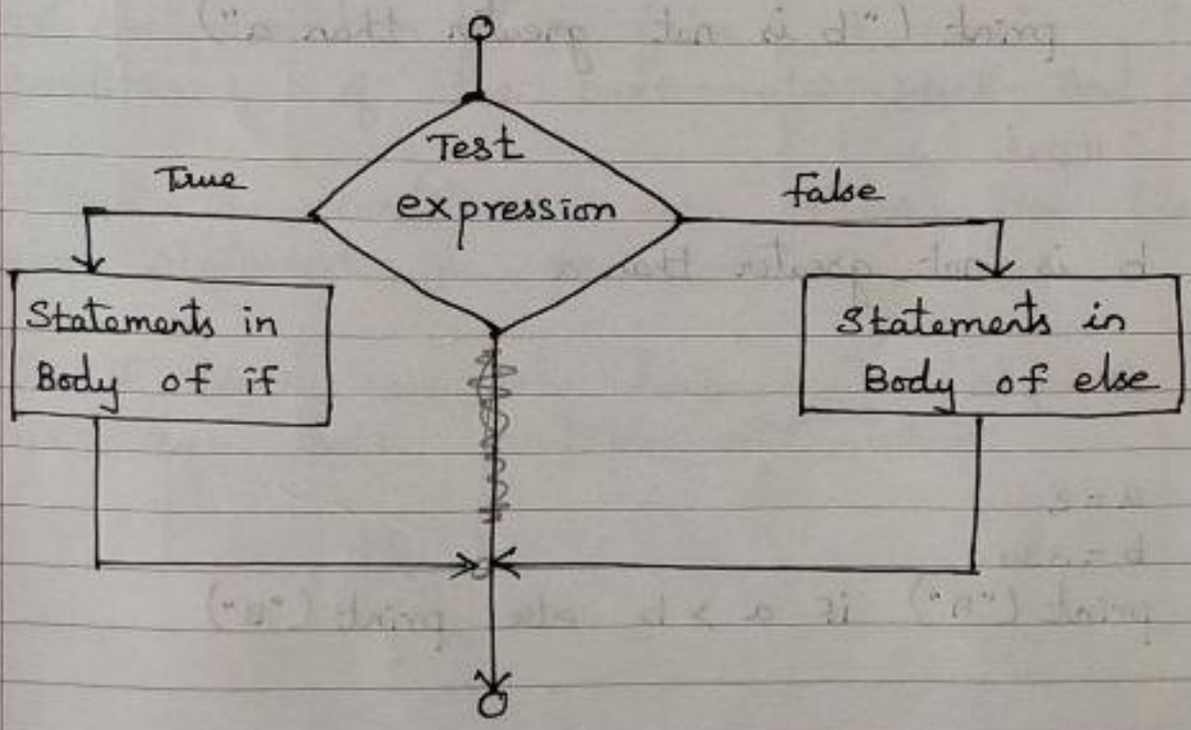
* if the condition is false, false statement block or else part is executed.

* Indentation is used to seperate the blocks.

Syntax

```
if test expression :
    body of if
else :
    body of else
```


Flowchart



Success
DATE:
PAGE:

Example

```
a = 330
b = 330
print ("A") if a > b else print ("=") if a == b else
                                        print ("B")
```

Output

=

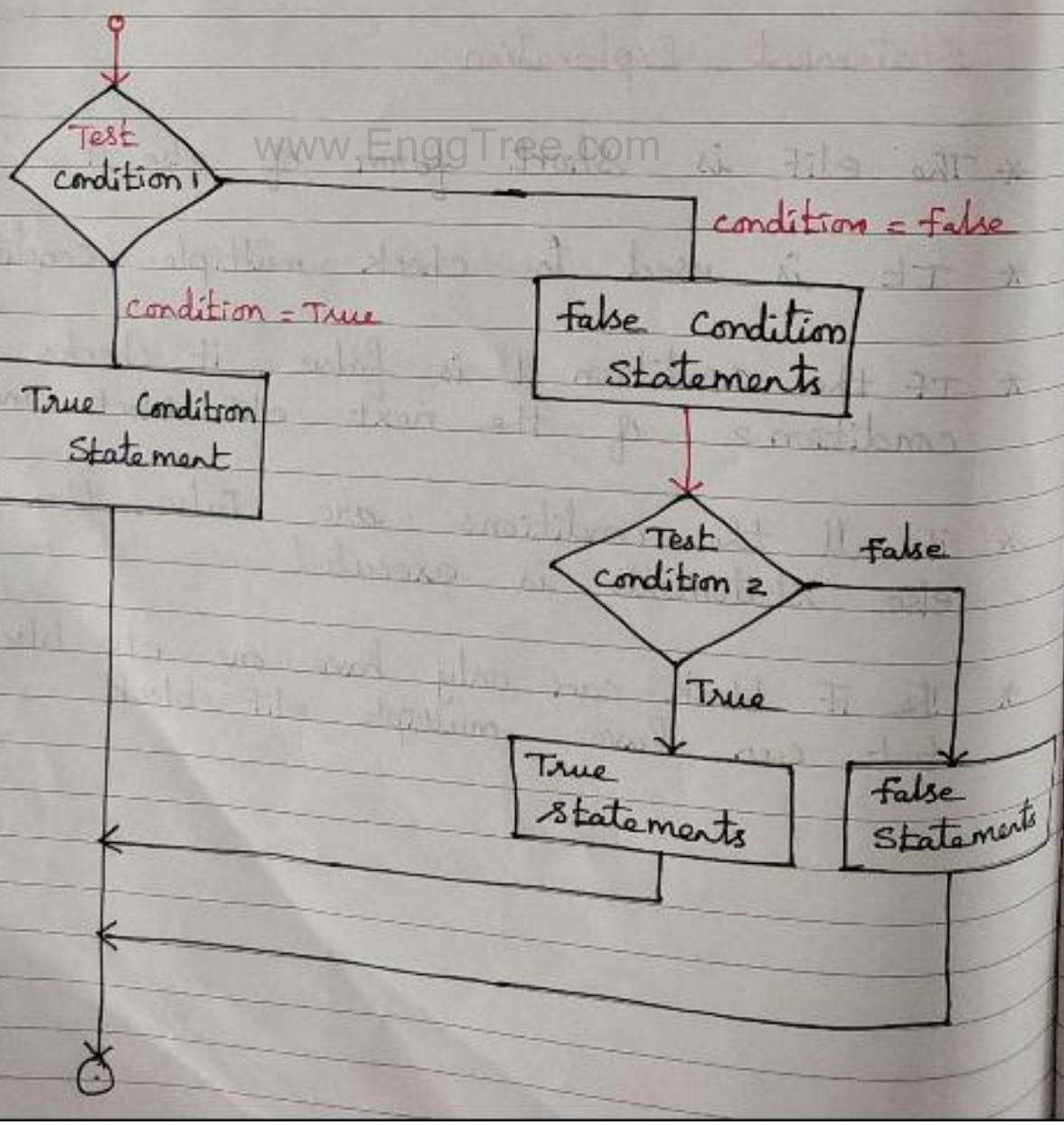
if..... elif .... else Statement (or) chained Conditional Statement Explanation

* The elif is short form of else if
* It is used to check multiple conditions
* If the condition 1 is false, it checks condition 2 of the next elif block and so on.
* If all the conditions are False, then the else statement is executed
* The if block can only have one else block, but can have multiple elif block.

Syntax

```
if Condition 1 :
    True Statement Block for condition 1
elif Condition 2:
    True Statement Block for condition 2
elif Condition 3 :
    True statement Block for condition 3
else
    False Statement Block
```

Flowchart



Example

```
num = 3
if num > 0 :
    print ("positive no")
elif  num == 0 :
    print ("zero")
else :
    print ("Negative number")
```

Output

```
positive no
```

Example


```
a = 200
b = 33
if b > a :
    print ("b is greater than a")
elif  a == b :
    print ("a and b are equal")
else
    print ("a is greater than b")
```

Output

```
a is greater than b
```

Multiple else statement on the same line

Program

```
a = 330
b = 330
print("A") if a>b else print("=") if a==b else print("B")
```

Output

=

The and keyword is a logical operator and is used to combine conditional

Program

```
a= 200
b= 33
c = 500
if a > b and c > a:
   print("Both conditions are True")
```


Output

Both conditions are True

The or keyword is a logical operator and is used to combine conditional statements

Program

```
a = 200
b = 33
c = 500
if a>b or a>c:
   print("At least one of the conditions is True")
```

Output

At least one of the conditions is True

Nested if

You can have if statements inside if statements, this is called nested if statements.

Program

```
X=41
if x>10:
  print ("Above ten,")
  if x>20:
    print ("and also above 20!")
  else
    print ("but not above 20")
```

Output

```
Above ten
and also above 20
```

The Pass Statement

if statement cannot be empty, but if ~~you~~ for some reason if statement with no content, put in the pass statement to avoid getting an error.

Program

```
a=33
b=200
if b>a:
  pass
```

Iteration

python allows a block of statements to be repeated as many times as long as the processor could support. This is generally called looping

A looping statement also influences the flow of control because it causes one or more statements or a block to be executed repeatedly.

* Looping is also called as repetition structure or iteration. An iteration structure allows the programmer to perform an action which is to be repeated until the given condition is True.

Break Statement

Statement :

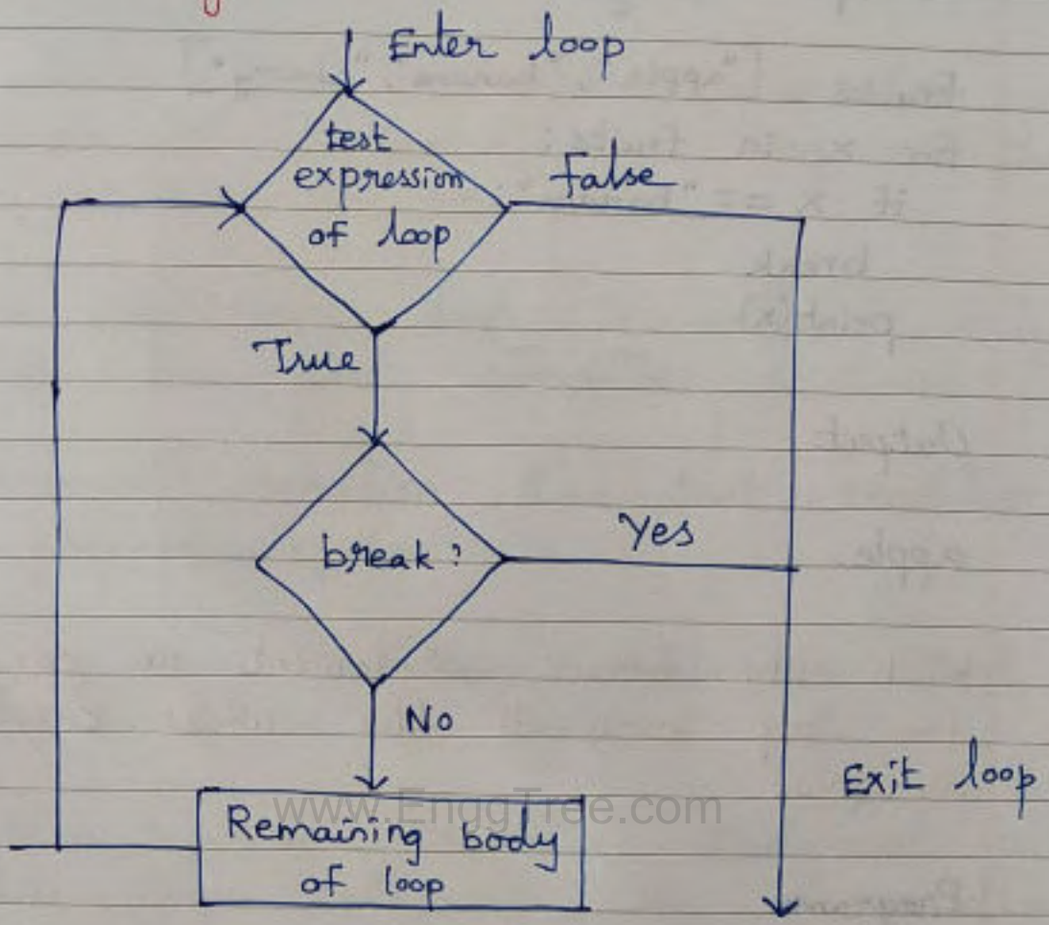
The break statement in python terminates the current loop and resumes execution at the next statement

Syntax.

```
break
```

DATE:
PAGE:

State Diagram



Example

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

Output

```
apple
banana
```

Example program

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

Output

apple.

With the break statement we can stop the loop even if the while condition is true

Program

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

Output

1

2

3

Success
DATE:
PAGE:

While loop

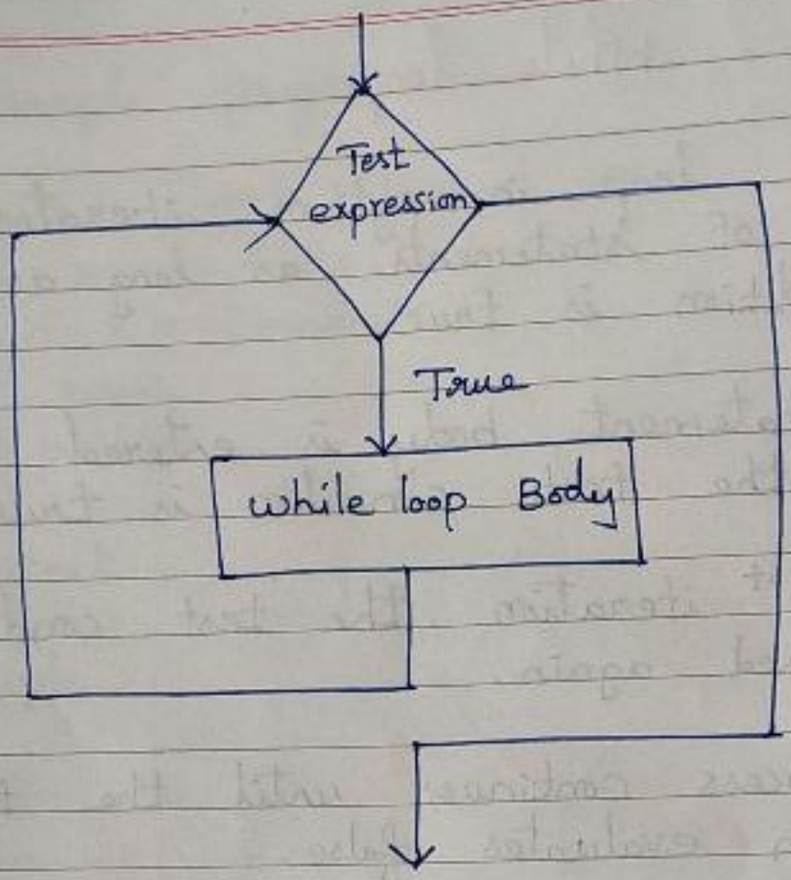
The while loop in python iterates over a block of statements as long as the test condition is true.

* The statement body is entered only when the test condition is true.
* After 1st iteration, the test condition is checked again.
* The process continues until the test condition evaluates false.

* python supports an else statement with a while loop statement. The else statement is executed when condition becomes false.

Syntax

```
While condition :
    Statement 1
    ....
    .....
    Statement n
```



With the while loop we can execute a set of statements as long as a condition is true

Program

```
i=1
while i<6:
  print(i)
  i+=1
```

Output

```
1
2
3
4
5
```

while loop with else

* Similar to for loop, while can also have optional else block.
* Else part is evaluated, if condition in the while loop evaluates false.
* The while loop can be terminated with break statement
* Hence else part runs if no break occurs and condition is false.

Syntax (while ... else loop)


```
while condition:
    Statement 1
    ....
    Statement n

Else:
    Statement 2
    ....
    Statement n
```

The else Statement

With the else statement we can run a block of code once when the condition no longer is true.

Print a message once the condition is false.

```
i=1
while i<6:
  print(i)
  i+=1
else
  print("i is no longer less than 6")
```

Output

```
1
2
3
4
5
i is no longer less than 6
```

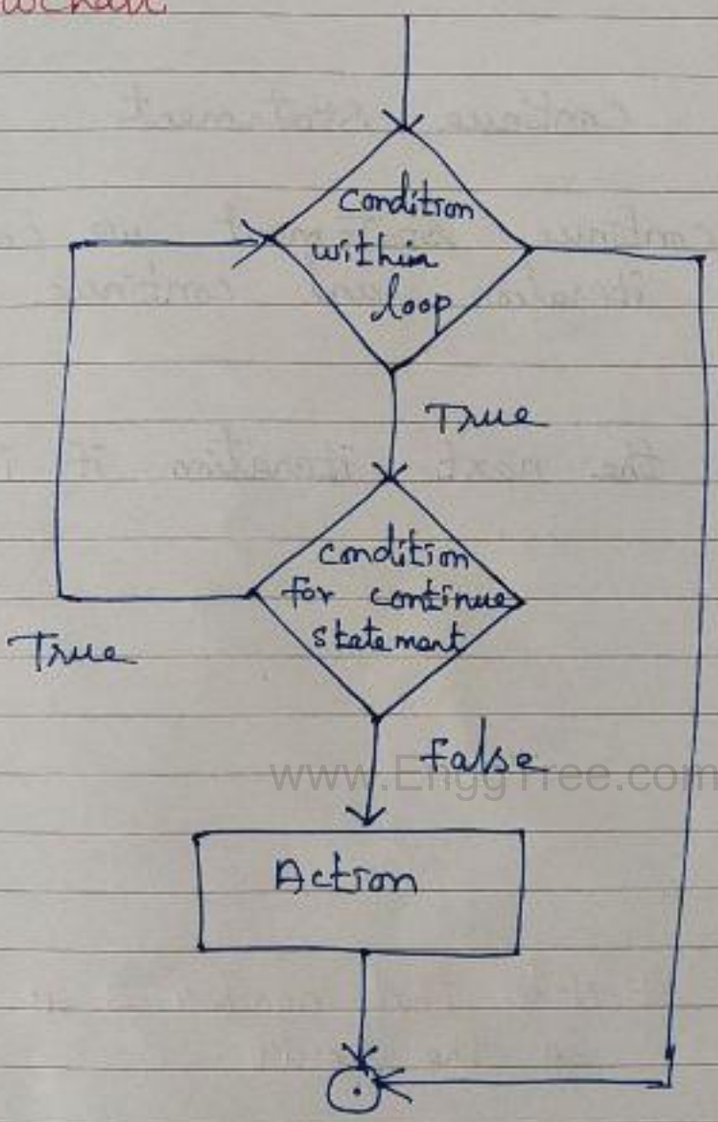

The Continue Statement

* The Continue Statement moves the control to the beginning of the while or for loop
* The continue statement rejects all the remaining statement in the current iteration of the loop.
* And moves the control back to the of the loop

Syntax

Continue

Flowchart



Continue Statement

* With the continue statement we can stop the current iteration of the loop and continue with the next

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    if x == "banana":
        continue
    print(x)
```

Output

apple
cherry

The Continue Statement

* With the continue statement we can stop the current iteration and continue with the next.

* Continue to the next iteration if i is 3

```
i = 0
while i < 6 :
  i += 1
  if i == 3 :
    continue
  print (i)
```


Output

```
1
2
4
5
6
```

Note that number 3 is missing in the result

The Pass Statement

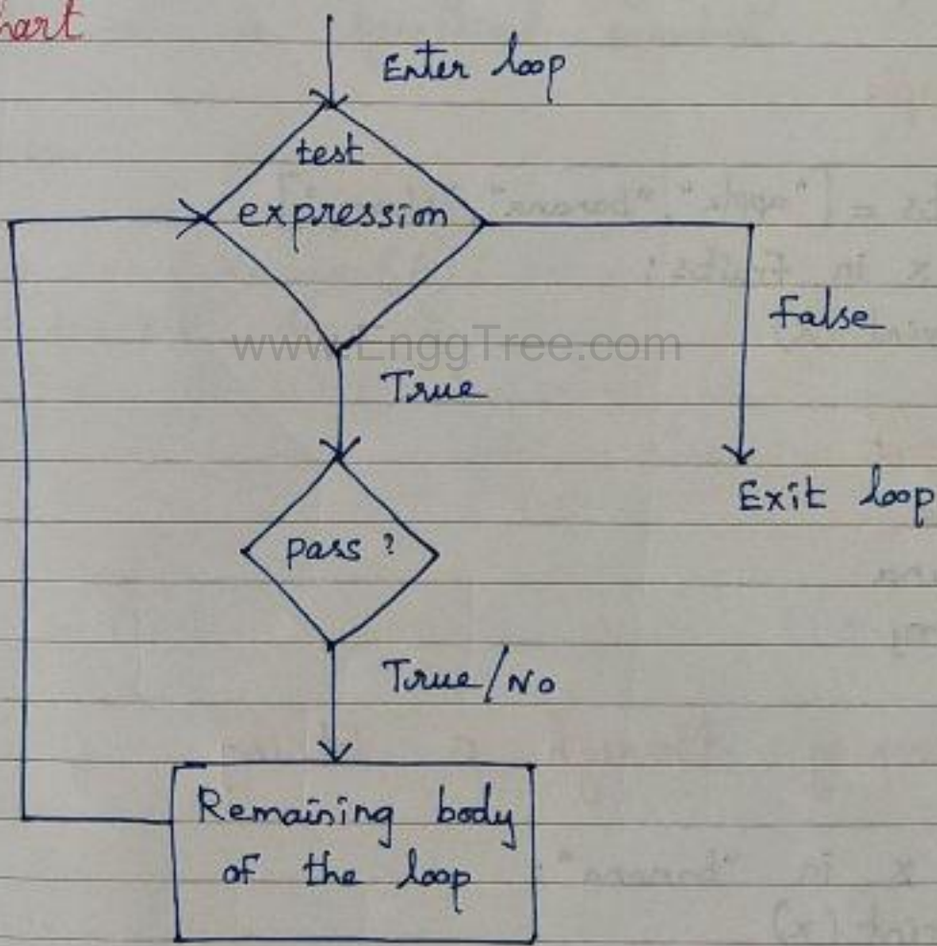
* The pass statement in python is used pass the control to the next statement, used in times where Syntax of the program has to be meet.
* The pass statement is a null operation it just passes the control.

* For loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.

Syntax

```
Pass
```

Flowchart



Example

```
for x in [0,1,2]:
    pass
```

For loop

* A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set or a string)

* With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc

Example

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Output

```
apple
banana
cherry
```


Looping through a String

```
for x in "banana":
  print(x)
```

Output

```
b
a
n
a
n
a
```

The range() function

* To loop through a set of code a specified number of times, we can use the range() function.

* The range() function returns a sequence of numbers, starting from 0 by default and increments by 1 (by default), and ends at a specified number.

Example

```
for x in range(6):
  print(x)
```

Output

```
0
1
2
3
4
5
```


* Note the range(6) is not the values of 0 to 6, but the values 0 to 5.

* The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2,6) which means values from 2 to 6 (but not including 6)

Example

```
for x in range (2,6):
    print (x)
```

Output

```
2
3
4
5
```

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range (2, 30, 3)

Increment the sequence with 3 (default is 1)

```
for x in range (2, 30, 3):
    print (x)
```

Output

```
2
5
8
11
14
17
20
23
26
29
```

Success
DATE:
PAGE:

Else in for loop

* The else keyword in a for loop specifies a block of code to be executed when the loop is finished.

Print all numbers from 0 to 5, and print a message when the loop has ended.

```
for x in range(6):
  print(x)
else
  print("Finally finished!")
```

Output


```
0
1
2
3
4
5
Finally finished
```

Break the loop when x is 3, and see what happens with the else block

```
for x in range(6):
  if x==3: break
  print(x)
else:
  print("Finally finished")
```

if the loop breaks the else block is not executed

Output

```
0
1
2
```

Nested Loops

* A nested loop is a loop inside a loop

* The "inner loop" will be executed one time for each iteration of the "outer loop"

Example

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print (x, y)
```


Output

```
red   apple
red   banana
red   cherry
big   apple
big   banana
big   cherry
tasty apple
tasty banana
tasty cherry
```

Success
DATE
PAGE

Fruitful Functions

* Functions that return value are called fruitful functions.

* In many programming languages a function that doesn't returns any value is called procedure.

* The return fruitful function is followed by an expression which is evaluate.

* Its result is returned to the caller as the fruit of calling the function hence the term fruitful.

Input the value → Fruitful is → Return the result

Example

Let us create a simple function called add

* The add function takes two numbers as parameters and returns the result of adding 2 nos.

* Thus add() is a fruitful function.

Example

input the 2 nos → Adder → Return the addition of 2 nos

Example

```
def add(a,b):
    c=a+b
    print("Sum of digits:", c)

add(10,2)
```

Example

```
def add(a,b):
    c=a+b
    print("Sum=", c)
    return c
add(10,2)
```

Example

```
def add(a,b):
    c=a+b
    print(c)
a=int(input("Enter the first number"))
b=int(input("Enter the second number"))
result=add(a,b)
if(result%2==0):
    print("Even")
else
    print("odd")
```

Output

```
Enter the first number:1
Enter the second number:1
2
```

TypeError: unsupported operand type(s) for %: NoneType and int

Example

```
def add(a,b):
    c=a+b
    print(c)
    return(c)
a=int(input("Enter the first number:"))
b=int(input("Enter the second number:"))
result=add(a,b)
if(result%2==0):
   print("Even")
else:
   print("odd")
```


Output

```
Enter the first number:1
Enter the second number:1
2
Even
```

Parameters in Fruitful function

* Parameter is the input data that is sent from one function to another.
  The paramters are of 2 types.

1. Formal parameter
2. Actual parameter

Formal Parameters

This parameter is defined as a part of the function definition

* The actual parameter value is received by the formal parameter.

Actual parameter

* This parameter is defined in the function call

* The actual parameter is passed to the formal parameter that is defined in the function.

Example program to show actual and formal parameter.

```
def cube(x):                    # x is the formal parameter
    return x * x * x
a = input("Enter the number:")
b = cube(a)                     # a is the actual parameter
Print("Cube of the given no:", b)
```

Output

```
Enter the number: 2
Cube of the given no: 8
```

Success
DATE:
PAGE:

Explanation

* Cube(a) is the function name and 'a' is the actual parameter
* This cube(a) invokes the function definition at statement number 3.
* The parameter in function definition is called the formal parameter.
* The value of 'a' is collected by formal parameter x.

Scope of the Variable

There are 2 scope of variable

1. Local variable with local scope
2. Global variable with global scope

* A variable in the program can be either local or global variable.
* Global variable is a variable that is declared in the main program.
* A local variable is a variable that is declared within the function.

Example

```
def f():
    s = "inside function"        # local variable
    print(s)
s = "Outside Function"           # Global variable
f()
Print(s)
```

Output

inside function
Outside function

Function Composition

* When the function is called within another function, it is called function composition.

* Thus function composition is a way of combining functions such that the result of each function is passed as the argument of the next function.

Syntax

f(g(x))

Where x is the argument of g
The result of g is passed as the arg of f
Thus the result of composition is the result of f

Success
DATE:
PAGE:

Example

* To find the distance between two points using function composition.

finding distance between two points

$$d = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$$

$(x_2, y_2) = (7, 8)$

$(x_1, y_1) = 3, 2$

Program

```
import math
x1 = int (input("Enter x1 : "))
Y1 = int (input("Enter Y1 : "))
X2 = int (input (" Enter X2 : "))
Y2 = int (input (" Enter Y2 : "))
distance = math. Sqrt (((X2-X1) ** 2) + ((Y2-Y1) ** 2))
print (" Distance", distance)
```

Output

```
Enter X1 : 3
Enter Y1 : 2
Enter X2 : 7
Enter Y2 : 8
Distance = 7.211102550927978
```

Strings

* Strings in python are surrounded by either single quotation marks or double quotation marks.

* 'hello' is the same as "hello".

* You can display a string literal with the print() function.

Example

```
print ("Hello")
print ('Hello')
```


Output

```
Hello
Hello
```

Assign String to a Variable

```
a = "Hello"
print (a)
```

Output

```
Hello
```

Multiline Strings

* You can assign a multiline string to a variable by using three quotes:

```
a="""Grace college of Engineering,
Thoothukudi
Mullakadu
Tamilnadu
"""
Print(a)
```

Output

```
Grace college of Engineering
Thoothukudi
Mullakadu
Tamilnadu.
```


String length

To get the length of a string, use the len() function.

```
a="Hello, World!"
print(len(a))
```

Output

12

String Slice

python provides Slice operation [ ] and [:] are used extract substring from the string.

* In python, the indexing of the characters start from 0, therefore the index value of the first character is 0.

PAGE:

Syntax

Sample string [start : stop : step]

Start and step are optional

Programs

```
b = "HelloWorld"
print (b[2:5])
```

Output

```
llo
```


Slice from the start

By leaving out the start index, the range will start at the first character.

Programs

```
b = "Grace College"
print (b[:7])
```

Output

```
Grace C
```

DATE:
PAGE:

Slice to the End

By leaving out the end index, the range will go to the end.

Program

```
b = "Grace College"
print (b[3:])
```

Output

```
ce College
```

Negative Indexing

Use negative indexes to start the slice from the end of the string.

Program

```
b = "Grace College"
print (b[-5:-2])
```

Output

```
lle
```

Upper Case

```
a = "Hello, World!"
print (a.upper())
```

Output

```
HELLO, WORLD!
```

Lower Case

```
a = "Hello, World!"
print (a.lower())
```

Output

hello, world!

Remove Whitespace

* Whitespace is the space before and/or after the actual text and very often you want to remove this space.

* The strip() method removes any whitespace from the beginning or the end.

~~Output~~ Program

```
a = " Hello, World! "
print (a.strip())
```

Output

Hello, World!

Replace String

```
a = "Hello, World!"
Print (a.replace ("H", "J"))
```

Output

Jello, World!

Success
DATE:
PAGE:

Split String

```
a = "Hello, Grace, world!"
b = a.split(", ")
print(b)
```

Output

```
['Hello', 'Grace', 'world!']
```

String Concatenation

```
a = "Hello"
b = "World"
c = a+b
print(c)
```


Output

```
HelloWorld
```

To add a space between them and a " "

Program

```
a="Hello"
b= "World"
c=a+" "+b
print (c)
```

Output

```
Hello World
```

String Module

* The string module provides additional tools to manipulate strings

* This module contains a number of functions to process standard python strings.

Program

```
import string
text = "Grace college of Engineering"
print ("Upper :", text.upper())
print ("Lower:", text.lower())
print ("Split:", text.split())
print ("replace:", text.replace ('college', 'institute'))
print ("count :", text.count ("n"))
```

Output

```
Upper : GRACE COLLEGE of ENGINEERING
Lower : grace college of engineering
Split : ['Grace', 'College', 'of', 'Engineering']
replace : Grace Institute of Engineering
Count : 3
```

Explanation

* Modules in python are simply the python files with .py extension which implements of functions.

* The modules are imported from the library using the import command.

* Once the String module is imported, the various string functions can be used.

* Thus every function in an module is accessed with an object.

* Example text.split()

* Here text is the object, split is the function connected with the "." operator.

List as an array

* Array and list, both are used in python to store data.

* Both can be indexed and iterated through

* The difference between a list and an array is the functions that you can perform on them.

Standard Library

```
From array import *
```

Array Construction

```
identifier_name = array (typecode, [initializer])
```

Typecode

'b' = signed integer 1 byte
'B' = unsigned integer 1 byte
'i' = Signed integer 2 byte
'I' = unsigned integer 2 byte

'f' = floating point 4 bytes
'd' = floating point 8 bytes

Create an array

```
from array import*
a=array('i',[1,2,3,4,5])
for i in a:
    print(i)
```

Output

```
1
2
3
4
5
```


Individual element can be called using index no

```
>>> from array import*
>>> a=array('i',[2,3,4,5,6])
>>> a[1]
3
>>> a[1]+a[2]
7
```

Sample methods of array objects

append()
insert()
extend()
index()
pop()
remove()
reverse()
to String()

Add new item to an array

append(), extend(), insert() or from list() methods are used to add new items to an existing array.

Program

```
from array import*
a=array('i',[2,3,4,5])
print(a)
a.insert(0,1)
print(a)
```

Output


```
array('i',[2,3,4,5])
array('i',[1,2,3,4,5])
```

Program

```
>>> from array import*
>>> a=array('i',[2,3,4,5])
>>> print(a)
array('i',[2,3,4,5])
>>> a.insert(0,1)
>>> print(a)
array('i',[1,2,3,4,5])
>>> a.append(6)
>>> print(a)
array('i',[1,2,3,4,5,6])
>>> b=array('i',[7,8,9])
>>> a.extend(b)
>>> print(a)
array('i',[1,2,3,4,5,6,7,8,9])
```

```
>>> c = [2,4,6]
>>> a.fromlist(c)
>>> print(a)
array('i', [1,2,3,4,5,6,7,8,9,2,4,6])
```

Removing an entry from array

```
>>> from array import*
>>> a = array('i', [4,6,2,9])
>>> a.remove(2)
>>> print(a)
array('i', [4,6,9])
>>> a.pop()
9
>>> print(a)
array('i', [4,6])
```


Thus we use list as an initializer while constructing an array.

ie) identifier_name = array(typecode, [initializer])

* Depending on our requirement we make use of list as an array by using the array concept.

* These are the Various ways of representations of list as an array.

Success
DATE:
PAGE:

Illustrative programs

① Sum an array of numbers

```
a = [0,0,0,0,0,0,0,0]
Sum = 0
n = int(input("Enter a number:"))
print("Enter n numbers:")
for i in range(0,n):
    a[i] = int(input())
for i in range(0,n):
    Sum = Sum + a[i]
print('Sum=', sum)
```

output

```
Enter a number : 4
Enter n number :
5
4
3
2
Sum = 14
```

	0	1	2	3	4	5	6
a	5	4	3	2			

Sum = Sum + a[i]

Sum = 0+5 = 5 ⇒ Sum = 5

Sum = 5+4 = 9 ⇒ Sum = 9

Sum = 9+3 = 12 ⇒ Sum = 12

Sum = 12+2 = 14 ⇒ sum = 14

PAGE

(2) Calculate GCD (Greatest common Divisor) using recursive function

Example: 12, 14

12 = 1, 2, 3, 4, 6, 12
14 = 1, 2, 7, 14

Find common divisor : 1, 2
Greatest No is : 2

Example

60, 72

a = b x Q + R

72 = 60 x 1 + 12 (a%b)

60 = 12 x 5 + 0

12 = 0 (a=b) when the b value is 0

b = 0, return a GCD = a = 12

Program

```
def computegcd (a,b):
if b==0:
  return a
else:
  return computegcd (b,a%b)
num1 = int (input ("Enter the first number:"))
num2 = int (input ("Enter the second number:"))
gcd = computegcd (num1, num2)
print ("GCD =", gcd)
```

Output

```
Enter the first number : 60
Enter the second number : 72
GCD = 12
```

③ Exponentiation of a number (e^n)

```
import math
num = int (input (" Enter a number :"))
ex = math.exp(num)
print (" Exponentiation =", ex)
```

Output


```
Enter a number : 3
Exponentiation = 20.085536923187668
```

④ Finding Square root of a number

```
n = int (input (" Enter a number :"))
x = int (input (" Number of times to calculate
                the approximation :"))
approx = 0.5 * n
for i in range(x):
    betterapprox = 0.5 * (approx + n / approx)
    approx = betterapprox
print (" Square root =", betterapprox)
```

Output

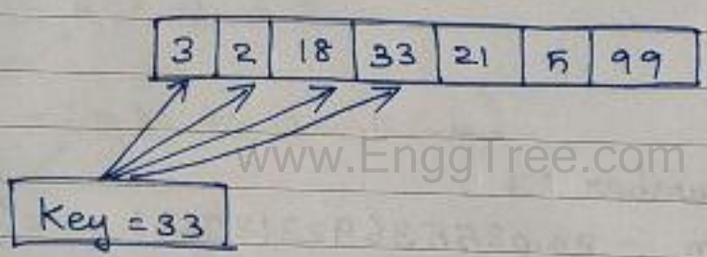
```
Enter a number : 10
Number of times to calculate the approximation : 3
Square root = 3.162319422150883
```

PAGE:

⑤ # Linear Search

* Searching a key value in an array
* Linear search is the simplest searching technique.
* The search begins at one end of the list and searches for the required element one by one until the element is found or till the end of the list is reached.

Example



Program

```
a = [0,0,0,0,0,0,0,0,0,0]
found = 0
n = int (input ("Enter a number :"))
print ("Enter n numbers :")
for i in range (0,n):
    a[i] = int (input())
key = int (input ("Enter a key to be searched:"))
for i in range (0,n):
    if key == a[i]:
        print ("Key found")
        found += 1
    else:
        continue
IF (found == 0):
    print ("Key not found")
```

DATE:
PAGE:

Output

```
Enter a number: 5
Enter n numbers:
1
2
3
4
5
Enter a key to be searched: 4
key found
```

Binary Search

* Binary Search requires the first to be sorted in ascending order.

* The binary search algorithm begins by comparing the element that is present at the middle of the list.

* If there is a match then the search ends and the location of the middle element is returned.

* IF there is a mismatch with the middle element and the search element is less than the middle element then first part of the list is searched otherwise second part of the list is searched.

0	1	2	3	4	5	6	7	8
9	13	25	39	42	54	67	74	81

Key = 67, $mid = \frac{0+8}{2} = 4$

0	1	2	3	4	5	6	7	8
9	13	25	39	42	54	67	74	81

Compare key with mid element $67 \neq 42$

$67 > 42$, So search 2nd list

5	6	7	8
54	67	74	81

Key = 67, $mid = \frac{5+8}{2} = 6$

5	6	7	8
54	67	74	81

Compare key with mid element
(6th element)

$67 = 67$
So search completed.

The key element 67 is found in 6th location

Success
DATE:
PAGE:

Program

```
def binary_search (item_list, item):
    first = 0
    last = len(item_list)-1
    found = false
    while (first <= last and not found):
        mid = (first + last) //2
        if item_list [mid] == item:
            found = True
        else:
            if item < item_list [mid]:
                last = mid - 1
            else
                first = mid+1
    return found
print (binary_search ( [1,2,3,5,8] ,6))
print ( binary_search( [1,2,3,5,8] ,5))
```


Output

```
False
True
```

UNIT - IV

LISTS, TUPLES, Dictionaries

List : list operations, list slices, list methods, list loop, mutability aliasing, cloning lists, list parameters. Tuples: tuple assignment, tuple as return value, Dictionaries: operations and methods, advanced list processing list comprehension; illustrative programs simple sorting, histogram, students mark statement, Retail bill preparation

List

List are used to store multiple items in a single variable.

* List are created using square brackets

Create a list

```
>>>a = ["apple", "banana", "cherry"]

>>> print (a)
```

O/P

```
['apple'] 'banana', 'cherry']
```

List items

List items are ordered, changeable and allow duplicate values.

* List items are indexed, the first item has index[0], the second item has index[0], the second item has index[1] etc.

Ordered

When we say that lists are ordered it means that the items have a defined order, and that order will not change.

* If you add new items to a list, the new items will be placed at the end of the list.

Changeable

The list is changeable, meaning that we can change, add and remove items in a list after it has been created.

Allow Duplicates

Since lists are indexed, lists can have items with the same value.

```
this list = ["apple", "banana", "cherry", "apple"]
print (thislist)
```

Output

```
['apple', 'banana', 'cherry', 'apple']
```

List Length

* To determine how many items a list has, use the len() function.

* list = ["apple", "banana", "cherry"]
  print (len (list))

Output

3

List Items - Data Types

List items can be of any datatype

```
List1 = ["apple", "banana", "cherry"]
List2 = [1, 5, 7, 9, 3]
List3 = [True, False, False]

print (List1)
print (List2)
print (List3)
```

Output

```
['apple', 'banana', 'cherry']
[1, 5, 7, 9, 3]
[True, False, False]
```

A list can contain different data types

* A list with strings, integers and boolean values.

```
List1 = ["abc", 34, True, 4.0, "Male"]
print (List1)
```

Output

['abc', 34, True, 4.0, 'male']

Access Items

* List items are indexed and you can access them by referring to the index number

```
list = ["apple", "banana", "cherry"]
print (list [1])
```

Output

banana

Negative Indexing

Negative indexing means start from the end

* -1 refers to the last item
  -2 refers to the second last item

```
this = ["apple", "banana", "cherry"]
print (this [-1])
```

output
cherry

Range of Indexes

* You can specify a range of indexes by specifying where to start and where to end the range

* When specifying a range, the return value will be a new list with the specified items.

```
list = ["apple", "banana", "cherry", "orange", "kiwi",
        "melon", "mango"]
print (list [2:5])
```

Output

['cherry', 'orange', 'kiwi']

Change Item Value

* To change the value of a specific item, refer to the index number

```
list = ["apple", "banana", "cherry"]
list [1] = "blackcurrant"
print (list)
```

Output

['apple', 'blackcurrant', 'cherry']

Insert Items

* To insert a new list item, without replacing any of the existing values, we can use the insert() method.

* The insert() method inserts an item at the specified index.

```
list = ["apple", "banana", "cherry"]
list.insert(2, "watermelon")
print(list)
```

Output

```
['apple', 'banana', 'watermelon', 'cherry']
```

Extend List

* To append elements from another list to the current list, use the extend() method.

```
list1 = ["apple", "banana", "cherry"]
list2 = ["mango", "pineapple", "papaya"]
list2.extend(list1)
print(list2)
```

Output

```
['mango', 'pineapple', 'papaya', 'apple', 'banana', 'cherry']
```

Remove Specified item

```
list = ["apple", "banana", "cherry"]
list.remove ("banana")
print (list)
```

Output

```
['apple', 'cherry']
```

The pop() method removes the specified index.

```
list = ["apple", "banana", "cherry"]
list.pop(1)
print (list)
```


Output

```
['apple', 'cherry]
```

The del keyword also removes the specified index.

```
list = ["apple", "banana", "cherry"]
del list[0]
print (list)
```

Output

```
['banana', 'cherry']
```

Clear the List

The clear() method empties the list

* The list still remains but it has no content.

```
list = ["apple", "banana", "cherry"]
list.clear()
print(list)
```

Output

```
[]
```


Loop through a list

You can loop through the list items by using a for loop.

Print all items in the list, one by one.

```
list = ["apple", "banana", "cherry"]
for x in list:
  print(x)
```

Output
apple
banana
cherry

Expression

The expression is the current item in the iteration but it also the outcome, which you can manipulate before it ends up like a list item in the new list.

Set the Values in the new list to uppercase.

```
Fruits = ["apple", "banana", "cherry", "kiwi",
          "mango"]
newlist = [x.upper() for x in Fruits]
print (newlist)
```

Output

['APPLE', 'BANANA', 'CHERRY', 'KIWI', 'MANGO']

Sort List Alphanumerically.

* List object have a sort() method that will sort the list alphanumerically ascending by default.

```
list = ["orange", "mango", "kiwi", "pineapple",
        "banana"]
list.sort()
print (list)
```

Reverse order

The reverse() method reverses the current sorting order of the elements.

```
list = ["banana", "orange", "kiwi", "cherry"]
list.reverse()
print (list)
```

Output

['cherry', 'kiwi', 'orange', 'banana']

Join Two Lists

* There are several ways to join, or concatenate, two or more lists in python

* One of the easiest ways are by using the + operator.

```
list = ["a", "b", "c"]
list1 = [1, 2, 3]
list3 = list1 + list
print (list3)
```

Output

```
[1, 2, 3, 'a', 'b', 'c']
```


List Slicing

* A subset of elements of a list is called a slice of list.

* Selecting a list slice is similar to selecting a character in a string

* We can easily access a range of items in a list of array the slicing operator (colon)

* Subsets and lists can be taking using the slice operator with two indices in square brackets. Separated by a colon ([] and [m:n])

Example

```
>>> birds = ['parrot', 'dove', 'duck', 'cuckoo']
>>> print (birds [2:4])

['duck', 'cuckoo']
```

Cloning List

* cloning list is to make a copy of the list itself, not just the reference.

* The easiest way to clone a list is to use the slide operator.

* Taking any slice of a list creates a new list.

* In this case the slice consists of the whole list

The changes made in the cloned list will not be reflected back in the original list.

```
>>> X = ['a', 'b', 'c']
>>> Y = X[:]
>>> print (x)
['a', 'b', 'c']
>>> print (Y)
['a', 'b', 'c']
>>> Y[0] = 'r'
>>> print (y)
['r', 'b', 'c']
>>> print (x)
['a', 'b', 'c']
```

List Parameters

* Passing a list as an argument actually passes a reference to the list, not a copy of the list.

* If the function modifies the list, the caller sees the change.

Delete function removes the first element from a list.

```
def delete(t):
  del t[0]
letters = ['a', 'b', 'c']
delete(letters)
print(letters)
```

output

```
['b', 'c']
```


The append method modifies a list, but the + operator creates a new list.

```
>>> t1 = [1,2]
>>> t2 = t1.append(3)
>>> print(t1)
[1,2,3]
>>> Print(t2)
None
```

```
t1 = [1,2,3]
t3 = t1 + [4]
print(t1)
[1,2,3]
print(t3)
[1,2,3,4]
```

Tuple Assignment

* python has a very powerful feature called the Tuple Assignment.

* This tuple assignment feature allows a tuple of variable on the right of an assignment to be assigned to the tuples of variable on the left.

Types

1. Tuple packing
2. Tuple unpacking

Tuple Packing

* In Tuple packing, the Values on the 'Right are packed' together is a tuple.

Example

```
>>> b = ("Grace", 18, "CSE")
```

Tuple Unpacking

In Tuple unpacking the values in a tuple on left are unpacked into the variables / names on the right

```
>>> b = ("bob", 18, "Mech")
>>> (name, age, studies) = b
>>> print(b)
('bob', 18, 'mech')
>>> print (name)
bob
>>> print (age)
18
>>> Print (studies)
mech
```

Using Tuple Assignment

```
(a, b) = (b, a)
```

* Each value is assigned to the respective variable.
* All expressions on the right side are evaluated before any assignment.
* Naturally the number of variables on the left and no of values on the right have to be same.
* Else it will throw an error.

Using a built in function divmod

```
t = divmod (7,3)
print(t)
```

(2,1)

Or use tuple assignment to store the elements separately.

```
>>> quot, rem = divmod (7,3)
>>> print (quot)
```

2

```
>>> print (rem)
```

1

Thus by using tuple, a function will return multiple values

Dictionaries

Dictionary is one of the datatype like string, list and tuple.

* Dictionaries are the combination of {key : value} pair.
* Here keys are unique within a dictionary while values may not be.

Dictionary Operations

Creating a Dictionary

Creating a dictionary is as simple as placing item inside curly braces {} separated by comma.

* Each item must have a key and the corresponding value expressed as a {key: value}

Different ways of creating dictionary

```
mydict = { }
mydict = {1: 'apple', 2: 'apple'}
mydict = {'name': 'John', 1: [2,4,3]}
mydict = dict {1: 'apple', 2: 'ball'}
```

Accessing elements from a dictionary

* Access the items of a dictionary by referring to its key name, inside square brackets.

* The key can be used either inside square brackets or with the get() method.

Example to access an element

```
thisdict = { "brand": "Ford",
             "model": "Mustang",
             "year": 1964
           }
x = thisdict ["model"]
print (x)
```

Output

Mustang

There is also a method called get() that will give you the same result

```
thisdict = { "brand" : "Ford",
             "model" : "Mustang",
             "year" : 1964
           }
x = thisdict.get("model")
print(x)
```

Output
Mustang

Get Keys

The keys() method will return a list of all the keys() in the dictionary.

```
thisdict = { "brand" : "Ford",
             "model" : "Mustang",
             "Year" : 1964
           }
x = thisdict.keys()
print(x)
```

Output

```
dict_keys(['brand', 'model', 'year'])
```

Add a new item to the original dictionary and see that the keys list gets updated as well

```
car = { "brand" : "Ford",
        "Model" : "Mustang",
        "Year" : 1964
      }
X = car.keys()
print(x)
```

```
car["color"] = "white"
print(x)
```

output

```
dict_keys(['brand', 'model', 'year'])
dict_keys(['brand', 'model', 'year', 'color'])
```

Get Values

The values() method will return a list of all the values in the dictionary.

Example

```
thisdict = {"brand" : "Ford",
            "model" : "Mustang",
            "year" : 1964
           }
x = thisdict.values()
print(x)
```

Output

```
dict_values(['Ford', 'Mustang', 1964])
```

Make a change in the original dictionary and see that the values list gets updated as well

```
car = {"brand" : "Ford",
       "model" : "Mustang",
       "year" : 1964
      }
x = car.values()
print(x)
car["year"] = 2020
print(x)
```

output

```
dict_values(['Ford', 'Mustang', 1964])
dict_values(['Ford', 'Mustang', 2020])
```

Add a new item to the original dictionary and see that the values list gets updated as well.

```
car = { "brand" : "Ford",
        "model" : "Mustang",
        "year" : 1964
      }
x = car.values()
print(x)
car["color"] = "red"
print(x)
```

Output

```
dict_values(['Ford', 'Mustang', 1964])
dict_values(['Ford', 'Mustang', 1964, 'red'])
```

Get Items

The items() method will return each item in a dictionary, as tuples in a list

```
thisdict = { "brand" : "Ford",
             "model" : "Mustang",
             "year" : 1964
           }
x = thisdict.items()
print(x)
```

Output

```
dict_items([('brand', 'Ford'), ('model', 'Mustang'),
            ('year', 1964)])
```

Removing items

There are several methods to remove items from a dictionary

The pop() method removes the item with the specified key name.

```
thisdict = {"brand": "Ford",
            "model": "Mustang",
            "year": 1964
           }
thisdict.pop("model")
print(thisdict)
```


Output

```
{'brand': 'Ford', 'year': 1964}
```

The popitem() method removes the last inserted item

```
thisdict = {"brand": "Ford",
            "model": "Mustang",
            "year": 1964
           }
thisdict.popitem()
print(thisdict)
```

Output

```
{'brand': 'Ford', 'model': 'Mustang'}
```

The del keyword removes the item with the specified key name.

```
thisdict = {"brand": "Ford",
            "model": "Mustang",
            "year": 1964
           }

del thisdict ["Model"]
print (thisdict)
```

Output

```
{'brand': 'Ford', 'Year': 1964}
```


The del keyword can also delete the dictionary completely

```
thisdict = {"brand": "Ford",
            "Model": "Mustang,
            "Year": 1964
           }
del thisdict
```

The clear() method empties the dictionary

```
thisdict = {
            "brand": "Ford",
            "model": "Mustang",
            "year": 1964
           }
thisdict.clear()
print (thisdict ())
```

Output

```
[]
```

Advanced list processing

The concept of list comprehensions is used to construct lists in an simpler manner by reducing the line of code.

* List comprehension executes much faster than for loop

* List comprehension is equivalent to for loop and if condition.

To display even numbers between 0-20

Without list comprehension

```
x=[ ]
for i in range (20):
    if i%2==0:
    x.append(i)
print(x)
```

Output

0, 2, 4, 6, 8, 10, 12, 14, 16, 18

With List Comprehension

```
evennos=[n for n in range(20) if n%2==0]
print(evennos)
```

Output

[0, 2, 4, 6, 10, 12, 14, 16, 18]

Thus list comprehension makes code simpler.

Illustrative Programs

Insertion Sort

* The insertion sort method sorts a list of elements by inserting each successive elements in the previously sorted sublist
* It performs n-1 passes.
* for pass i through n-1, insertion sort ensures that the elements in position 0 through p are in sorted order.
* In pass P, we move the p^{th} element to left until its correct place is found among the first element.

Program

```
def main():
    a=[]
    n=int(input("Enter list size:"))
    for i in range(n):
        a.append(int(input()))
    print(a)
    print(insertionsort(a))
main()


def insertionsort(a):
    for i in range(1, len(a)):
        currentvalue = a[i]
        pos=i
        while((pos>0) and (a[pos-1] > currentvalue)):
            a[pos] = a[pos-1]
```

```
        pos = pos-1
    is (pos != i):
        a[pos] = currentvalue
    return a
```

Output

```
Enter the list size : 5
12
35
24
01
33
[12, 35, 24, 1, 33]
[1, 12, 24, 33, 35]
```


Student Marks Statement

```
tamil = int (input ("Tamil : "))
English = int (input ("English : "))
Maths = int (input ("Maths : "))
Science = int (input ("Science : "))
Social = int (input ("Social : "))
total = tamil + English + Maths + Science + Social
print ("Total :", total)
avg = total /5
print ("Average :", avg)
if (tamil >= 50 and english >= 50 and maths >= 50
   and Science >= 50 and Social >= 50):
   print ("Result : Pass")
   else:
      print ("Result : fail")
```

Output

Tamil : 65
English : 32
Maths : 89
Science : 90
Social : 100
Total : 376
Average : 75.2
Result : FAIL

Histograms

* A histogram is a graph showing frequency distributions.
* It is a graph showing the number of observations within each given interval.

Create Histogram

* In Matplotlib, we use the hist function to create histograms.
* The hist() function will use an array of numbers to create a histogram, the array is sent into the function as an argument

For simplicity we use Numpy to randomly generate an array with 250 values, where the values will concentrate around 170, and the standard deviation is 10

A Normal Data Distribution by Numpy

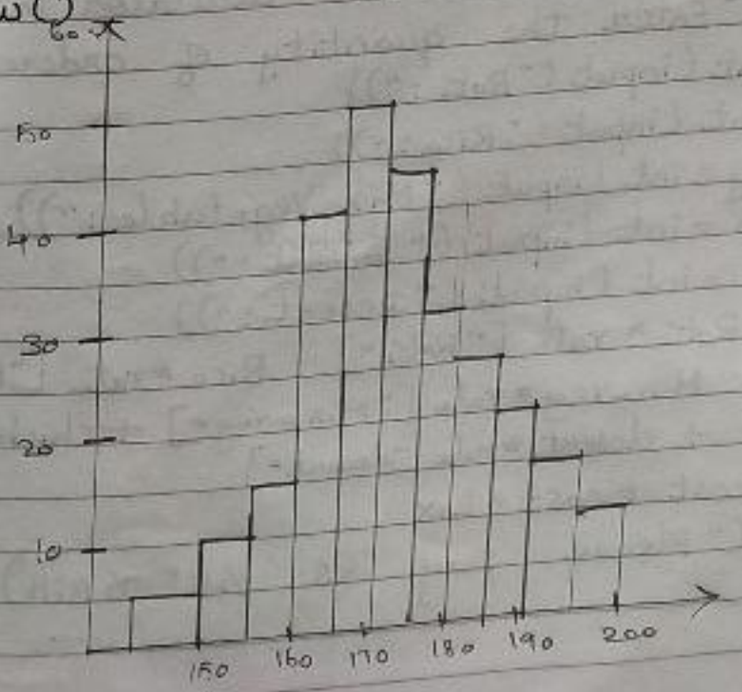
```
import numpy as np
x=np.random.normal(170,10,250)
print(x)
```

Output

```
169.04931476   173.10681503   162.65062026
165.2732938    179.28656775   162.78476117
 .              .              .
 .              .              .
 .              .              .
 .              .              .
 .              .              .
```

The hist() function will read the array and produce a histogram.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.normal(170,10,250)
plt.hist(x)
plt.show()
```



Matplotlib

Graphical representation that organizes a group of data points into specified range.

Input value → interval → count of value

↓

represent the data using bars

* Creating a histogram represents a visual representation of data distribution.
* Histogram can display large set of data

Retail Bill preparation

```
tax = 0.8
rate = {"Roti": 5, "Rice": 70, "Mix_veg": 100,
        "chicken": 200, "desert": 50}
print ("Restaurant Bill calculator \n")
print ("Enter the quantity of ordered item: \n")
Roti = int (input ("Roti: "))
Rice = int (input ("Rice: "))
Mix_veg = int (input ("Mix Vegetable: "))
chicken = int (input ("chicken: "))
desert = int (input ("Desert: "))
cost = Roti * rate ["Roti"] + Rice * rate ["Rice"] +
       Mix_veg * rate ["Mix_veg"] + chicken * rate ["chicken"]
       + desert * rate ["desert"]
Bill = cost + cost * tax
print ("Please pay Rs. %.f" % Bill)
```

Output

Restaurant Bill Calculator

Enter the quantity of ordered item:

Roti : 5
Rice : 2
Mix vegetable : 3
Chicken : 5
Desert : 2
please pay Rs. 837.000000

UNIT-5

Files and exceptions: text files, reading and writing files, format operator, command line arguments, errors and exceptions, handling exceptions, modules, packages
Illustrative programs: word count, copy file voter's age validation, Marks range validation (0-100).

Text files

Files

File is a named location on the system storage which records data for later access

* files enables persistant storage in a non-volatile memory ie) Hard disk.

Types of file

* Text file
* Binary file

Text files

* Text files are simple text files where are in human readable format, used for reading writing a content.
* Text files occupies more space.

Binary file

* Binary files contain binary data which is only readable by computer
* Binary files occupies lesser space than text mode.

File operations

* In python there is no need for importing external library to read and write files.
* python provides a built in function for opening, writing and reading files.

Opening a file

* A file can be opened using built-in open() function.
* It has the following methods like write, readline, read, writelines and close.
* The open function returns a python file object, which is an instance of the built in type file

Syntax
File object = open ("Filename", "accessmode")

Filename : It is a string value which contains the name of the file to access

Accessmode : It determines the mode (read, write, append etc)
* Parameter are read mode is the default file access mode.

The Various access mode

r → Opens a file for reading only.

r+ → Opens a file for both reading and writing

w → Opens a file for writing only

w+ → Opens a file for both writing and reading

a → Opens a file for appending. file pointer it at the end of the file.

a+ → Opens a file for both appending and reading.

Example

```
>>> f = open("E:/a.txt", "r")
>>> print (f.read())
Grace College of Engineering
```

Reading data from a file

* In order to read the content of a file we have to open a file in read mode (r)

There are several ways to read the text file in python.

Method - 1

read ([size]) → It returns the specified number of character from the file

* The size omitted it will read the entire content of the file.

Method - 2

readline() → It returns the nextline of the file

readlines() → It reads all the lines as a list of strings in files

Read [size]

The read [size] method reads a specified number of bytes from the open file and returns a string with the data that was read.

* If the size parameter is not specified then it reads and returns data upto end of file (EOF)

Syntax

Fileobject.read([size])

Here size is the optional numeric argument and returns a quantity of data equal to size.

a.txt

Grace college of Engineering

Python shell

```
>>> f=open("E:/a.txt","r")
>>> print(f.read(5))
Grace
```

readline()

* The readline() method is used to read individual lines of a file.
* It reads a file till the new line (\n) character is reached or end of file (EOF)

Syntax

```
fileobject.readline([size])
                    ↳ optional
```

a.txt

Grace college of Engineering
Mullakadu
Thoothukudi district.

Python Shell

```
>>> f=open("E:/a.txt","r")
>>> f.readline()
'Grace college of Engineering\n'
>>> f.readline()
'Mullakadu\n'
>>> f.readline()
'Thoothukudi district'
```

Readlines()

* The method readlines() returns a list of remaining lines on the entire file.
* These reading method returns empty string when end of file (Eof) is reached.

a.txt

```
Grace college of Engineering
Mullakadu
Thoothukudi district
```

Python Shell

```
>>> f=open("E:/a.txt","r")
>>> f.readlines()
['Grace college of Engineering\n', 'Mullakadu\n',
'Thoothukudi district']
>>> f.readlines()
[]
```

Writing a file

* The write of method is used to write a string or sequence of bytes (Binary files)
* It returns the number of characters written to the file.
* In order to write data to a file. we need to open a file either in write 'w', append 'a' or exclusive creation 'x' mode.
* Similar to 'w' the file must not exist before opening.
* In writing (w) mode all the data in the file will be overwritten, if the file already exists in the memory.

Syntax

```
fileobject.write(str)
```

str is the content to be written in opened file.

a.txt

```
I like python
```

Python shell

```
>>> f=open("E:/a.txt","w")
>>> f.write("I like python")
13
>>> f.close()
```

The File opened in write mode 'w' will rewrite all the content in the file with new content.

* Once you have opened a file in write mode, you should properly close() to reflect the changes.

Closing a File

In close() method, a file object flushes all information and closes the file object

* No more updation can be performed in that particular file once they are closed
* It automatically closes a file, when the reference object of a file is reassigned to another file in python.
* unreferenced object were cleared using Python garbage collector.

Syntax: fileobject.close()

Format operators

* Python's interesting features is that format operator
* This operator used to print the output in a desired format as the print() statement.
* The % operator is the format operator It is also called as interpolation operator
* The % operator is also used for modulus
* The usage varies according to the operands of the operator.

Syntax

% <format expression> % (V1, V2, V3 ..... Vn)

or

% <format expression> % values

where V1 to Vn are variables that are formatted using the expression.

where values is a tuple with exactly the number of items specified by the format string.

Various Conversion type available in python

d → Signed integer decimal

i → Signed integer decimal

o → unsigned octal

u → unsigned decimal

c → Single character (accepts int or single character string)

r → String (converts any python obj using repr())

s → String (converts any python obj using str())

Formatting Strings Example program to show all the possible Number formatting.

```
print ('simple integer formatting : %d' %(42))

print ('Simple Float formatting : %f' %(3.141592653589793))

print ('Float with 5 digits and 2 digits after :
        %5.2f' % (3.141592653589793))
```

Output

```
Simple integer formatting : 42

Simple Float formatting : 3.141593

Float with 5 digits and 2 digits after : 3.14
```

Command Line Arguments

python fully supports creating program that can be run on the command line

python provides

1. System Module : To access command line arguments via sys.argv
2. getopt Module : That helps you parse (analyse) command line options and arguments.
   * for running complex program you need getopt module.

Sys module

```
import sys
print (sys.version)
```

Output

```
3.2.5 (default, may 15 2013, 23:06:03)
[MSC v.1500 32bit (Intel)]
```

Sys.version is used which returns a string containing the version of python interpreter with some additional information

DATE: PAGE: Success

Input and Output using sys

* The sys module provide variables for better control over input or output.
* we can even redirect the input and output to other devices.
* This can be done using three variables
  - → Stdin
  - → stdout
  - → Stderr

Stdin

Stdin → It can be used to get input from the command line directly.

* It is used for standard input. It internally calls the input() method.
* It also automatically adds '\n' after each sentence.

Program

```
import sys
print ("The length of argument is", len(sys.argv))
print ("file name of this program", sys.argv[0])
a = sys.argv[1]
b = sys.argv[2]
print ("Value of a is", a)
print ("Type of a is", type(a))
c = a+b
print ("Sum of a and b", c)
```

Output

```
The length of argument is 3
file name of this program hhh.py
Sum of a and b 2020
Value of a is 20
Type of a is <'class 'int'>
sum of a and b 50
```


Stdin

```
import sys
for line in sys.stdin:
    if 'q' == line.rstrip():
        break
    print (f'Input: {line}')
print ("Exit")
```

Output

```
Input : Grace college of Engineering
q
Exit
```

stdout

* A built-in file object that is analogous to the interpreter's standard output stream in python.

* stdout is used to display output directly to the screen console.

* Output can be of any form, it can be output from a print statement, an expression statement, and even a prompt direct for input.

* By default, streams are in text mode.

* In fact wherever a print function is called within the code, it is first written to sys.stdout and then finally on to the screen.

Program

```
import sys
sys.stdout.write ('Grace')
```

Output

Command prompt

```
c:\users\sony\Desktop\python stdout.py
Grace
```

Stderr

Whenever an exception occurs in python it is written to sys.stderr

Reference Count

sys.getrefcount() method is used to get the reference count for any given object

* This value is used by python as when this value becomes 0, the memory for that particular value is detected.

Program

```
import sys
a='Grace'
print (sys.getrefcount(a))
```

Output

```
c:\users\sony\Desktop> python count.py
4
```

Errors

Errors or mistakes in a program are often referred to as bugs

* They are almost the fault of the programmers.

* The process of finding and eliminating error is called debugging.

Errors can be classified into 3 major groups

* Syntax errors
* Runtime errors
* Logical error or semantic error

Syntax error

* Python will find these kinds of errors when it tries to parse your program and exit with an error message without running anything

* Syntax errors are mistakes in the use of the python languages

Common syntax errors include

* Leaving out a keyword
* putting a keyword in the wrong place.
* Leaving out a symbol such as a colon, comma or brackets.
* Misspelling a keyword
* Incorrect indentation
* Empty block.

Runtime errors

If a program is syntactically correct that is free of syntax error, it will be run by python interpreter.

* However program may exit unexpectedly during execution if it encounters a runtime error.

* Runtime errors are those which are not detected when the program was parsed but is only revealed when program runs

* when program comes to a halt because of runtime error we say that it has crashed.

Logical errors

* Logical errors or semantic errors are the most difficult to fix.

* They occur when the programs runs without crashing but produces an incorrect result.

* Logical errors are caused by a mistake in the programs.

* You won't get an error message because no syntax or runtime error has occurred.

Success
DATE
PAGE

More frequently these kinds of errors are caused by programmers carelessness

Some examples of mistakes which lead to logical error are.

* Using the wrong variable name.
* Indenting a block to the wrong level
* Using integer division instead of floating point division.
* Getting operator precedence wrong.
* Making mistake in a boolean expression.

To overcome logical error

You will have to find the problems on your own by reviewing all the relevant parts of your code.

Python Built in Exceptions

* illegal operations can raise exceptions
* They are plenty of built in exceptions in python that raised when corresponding errors occurs
* we can view all the built in exceptions using local built in functions

Python Built in Exceptions

1. Assertion error – Raised when user defines unexpected values.
2. Attribute error – Raised when attribute assignment or reference fails.
3. EOf Error – Raised when the input() function hits the end of file conditions
4. Floating point Error – Raised when a floating point operation fails
5. Import error – Raised when the imported module is not found.
6. Index error – Raised when index of a sequence is out of range.
7. Key error – Raised when a key is not found in a dictionary.
8. System Errors – Raised when interpreter detects internal error.

DATE
PAGE

Handling Exceptions

In python Exception can be handled by using try, except and finally statements.

Try → Raise → Except

Exception may occur → Raise the Exception → Catch if exception occurs

* Python has many built in exception which forces your program to output an error when something in it goes wrong.

* When these exception errors it causes the current process to stop and passes control to the calling process until it is handled.

* If not our program will crash.

* If never handled an error message is split out and our program come to a sudden, unexpected halt.

Catching Exceptions in python.

* In python exception can be handled using a try statement.

* A critical operation which can raise exception is placed inside the try clause and the code that handles exception is written in except clause.

program

```
try
    print(x)
except
    print("An exception occurred")
```

Since the try block raises an error, the except block will be executed.

Finally

The finally block, if specified, will be executed regardless if the try block raises an error or not

Program

The finally block gets executed no matter if the try block raises any errors or not

```
try:
    print(x)
except:
    print("Something went wrong")
finally:
    print("The try except is finished")
```

Output

```
Something went wrong
The try except is finished
```

Success
DATE
PAGE

Modules

Modules are pre written pieces of code that are used to perform common tasks like generating random number, performing mathematical operations etc.

* A module is a file with .py extension that has definitions of all functions and variables that would use in other programs.

* Modules are used to break down large programs into small manageable and organized files and it provides reusability of code.

Creating a module

The modules can be created as we want. Every python program is a module, that is every file saved as .py extension is a module.

Example

```
def add(a,b):
    result = a+b
    return result
```

mymodule.py

```
person1 = {"name": "John",
           "age": 36,
           "country": "Norway"
          }
```

Import the module named mymodule and access the person1 dictionary

```
import mymodule
a = mymodule.person1["age"]
print(a)
```

Output

36

Example

```
from math import pi
print("The value of pi is", +pi)
```

Output

The value of pi is 3.1415926

To import more than one item from a module, use a comma seperated list

```
from math import pi, sqrt
```

DATE
PAGE

DATETIME Module

* A date in python is not a datatype of its own, but we can import a module named datetime to work with dates as date objects.

* python display date in YYYY-mm-dd format.

* python datetime module handles the extraction and formatting of date and Time variables objects of the datetime type represent a date and a time in some timezone.

* Calendar date values are represented with the date class. All the attributes are accessed through the dot(.) operator with the date object.

Import the datetime module and display the current date

```
import datetime
x = datetime.datetime.now()
print(x)
```

Output

```
2022-02-22 10:00:08:947693
```

The date contains year, month, day, hour, minute, second and microsecond.

PAGE

The Strftime() Method

* The datetime object has a method for formatting date object into readable string.
* The method is called Strftime() and takes one parameter, format to specify the format of the returned string

Display the name of the month

```
import datetime
x = datetime.datetime(2018,6,1)
print(x.strftime("%B"))
```

Output
June.

MATH Module

* python provides many useful mathematical functions in a special math library.
* To access one of the functions we have to specify the name of the module and the name of the function separated by a dot.
* This format is called dot(.) notation

Success
DATE
PAGE

import math

This statement imports the entire math module methods into the current application

```
import math
print math.sqrt(100)
```

Output
10.0

```
import math
x = math.ceil(1.4)
y = math.floor(1.4)
print (x)
print (y)
```


Output
2
1

Packages

A package is a way of collecting related modules together within a single tree like hierarchy.

* It has a well organized hierarchy for easier access.

* The directory can contain sub directories and files where as a python package can have sub packages and modules.

* A package must contain a file named __init__.py in order for python to consider it as a package

* This file can be left empty.

Create a package

* Create a directory and name it with a package name.

* Keep sub directories (sub packages) and modules in it.

* Create __init__.py file in the directory

* The __init__.py file can be left empty but generally place the initialization code with import statements to import resources from a newly created package.