UNIT - I

ALGORITHM

↳ An algorithm is defined as step by step procedure for solving a problem.

↳ Well-defined computational procedure consisting of a set of instructions that takes some value or set of values, as input and produces some value or set of values, as output

Properties

↳ input specified

↳ Output specified

↳ Definiteness

↳ Effectiveness

↳ Finiteness

Qualities

↳ Time - The lesser is the time required, better is the algorithm

↳ Memory - The lesser is the memory required, better is algorithm

↳ Accuracy - Multiple algorithms may provide suitable or correct solutions to a given problem, better is the algorithm which give more accurate result than others.

Characteristics

↳ Well-ordered - Proper order has to be maintained.

2. Unambiguous operations :

Each step in an algorithm must be simple so that it can be translated directly into programming steps & for execution.

3. Effective computational operations:

Each step of algorithm must be achievable by computer

4. Input

An algorithm has zero or more inputs.

5. Has a result

An algorithm must generate result.

Example : Write an algorithm compute larger of two numbers.

Alg :

```
Step1 : Start
Step2 : Read numbers a & b
Step3 : If a>b then goto step 4,
             else goto step 5
Step 4 : print " a is greatest"
Step 5 : print " b is greatest"
Step 6 : Stop.
```

Logic.

```
1. Start
2. a,b
3. a>b
4.      a is greatest
     else
5.      b is greatest
6. stop.
```

Advantages.

↳ Easy to understand

↳ Programs can be easily developed

↳ Independent of programming lang

↳ Helps to debug the program logic

Disadvantages.

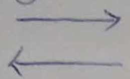
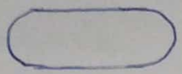
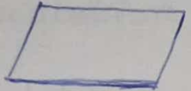
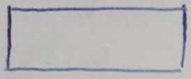
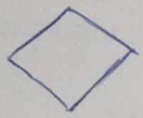
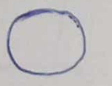
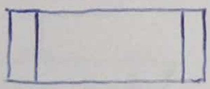
↳ No particular rule is available

↳ Involves more repetitions of Writing Work.

FLOWCHART

↳ defined as "graphical representation of logic for problem solving"

↳ Purpose of flowchart is "Making logic of program clear in a visual representation".

	Symbol	Symbol Name
1.		flowlines
2.		Terminal
3.		input/output
4.		Processing.
5.		Decision
6.		Connector.
7.		Subfunction

Rules for drawing a flowchart.

↳ Clear, neat, easy to follow

2. Have logical start and stop

3. Only one flow line should comeout from process symbol.

4. Only one flowline should enter in,
Two or three flowlines can leave from decision symbol

5. Only one flowline is used with Terminal symbol, either inflowline or outflowline.

6. Flowlines should not be intersect

7. Write preciously within the symbol.

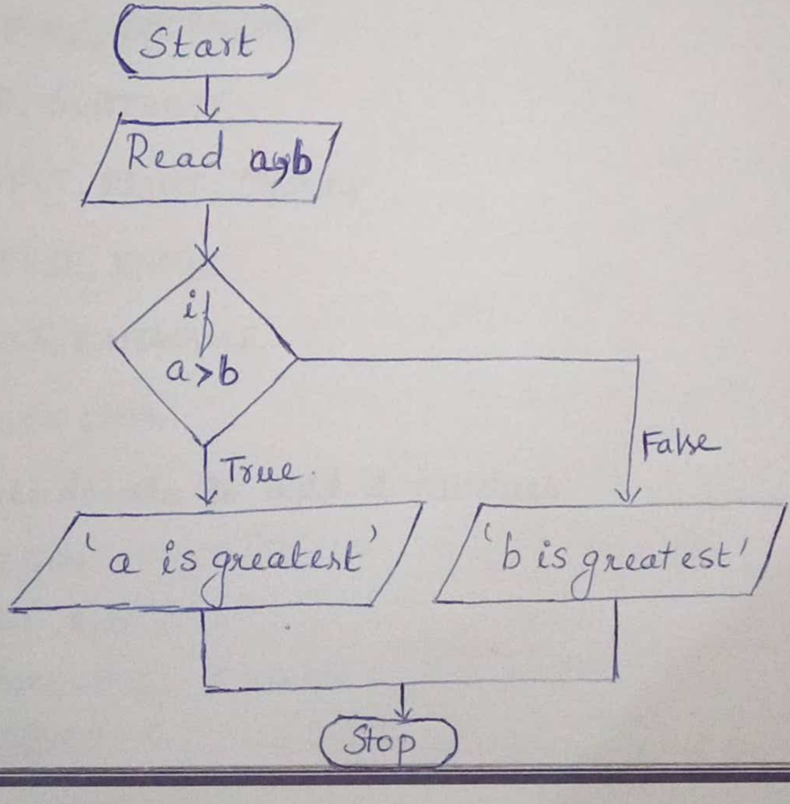
Advantages

1. Communication
2. Effective analysis
3. Proper documentation
4. Efficient coding.
5. Proper debugging.
6. Efficient program maintenance.

Disadvantages

1. Complex logic cannot be represented
2. Alterations and modifications are difficult
3. Difficult to reproduce
4. Cost is high.

Example : Draw flowchart to compute larger of 2 No's.



Logic.

1. Start ⬭
2. a,b ▱
3. a>b ◇
4. a is greatest ▱
else
5. b is greatest ▱
6. stop ⬭

PSEUDOCODE

↳ defined as short, readable and formally styled English languages used for explain an algorithm.

Guidelines for writing pseudocode.

- Write one statement per line
- Capitalize initial keyword.
- Indent to hierarchy.
- End multiline structure.
- Keep statements language independent.

Common keywords.

1. //
2. BEGIN, END.
3. INPUT, GET, READ
4. INITIALIZE.
5. COMPUTE, CALCULATE
6. ADD, SUBTRACT
7. OUTPUT, PRINT, DISPLAY
8. IF, ELSE, ENDIF
9. WHILE, ENDWHILE.
10. FOR, ENDFOR

Eg: Pseudocode to add 2 numbers.

```
BEGIN
GET a,b
CALCULATE  c=a+b
DISPLAY  c
END
```

Advantages:

- ↳ Independent of any language.
- ↳ Easy to translate to any programming lang
- ↳ Easily modified.

Disadvantages:

- ↳ Not provide visual representation
- ↳ No accepted standards.
- ↳ cannot be compiled nor executed.
- ↳ more difficult to follow.

BUILDING BLOCKS OF AN ALGORITHM

Statements : Single action

Input statement

Process statement

Output statement.

State:

Transition from one state to another under specified condition within a time.

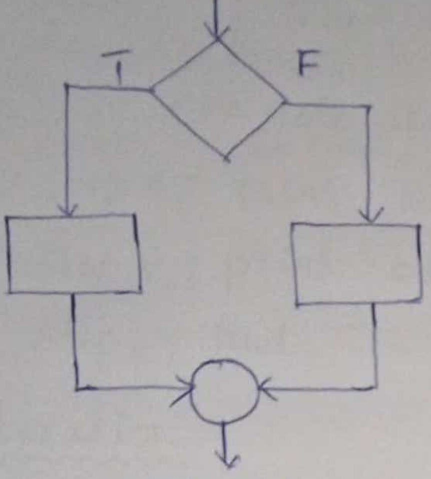
Control flow:

The process of executing the individual statements in given order.

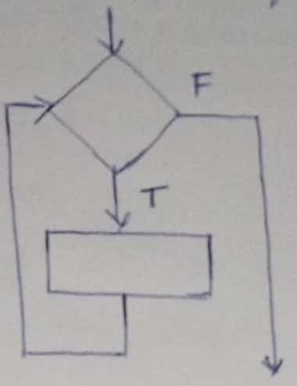
1. Sequence.
2. Selection.
3. Iteration.

Function:

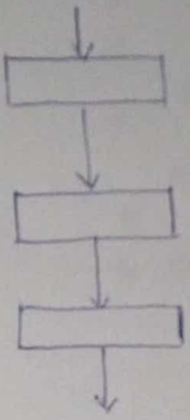
A set of staliments that perform a specific task.



Selection



Iteration



Sequence

Sequence.

Instructions are executed one after another.

Eg: Algorithm to calculate Simple Integer.

```
Step1 : Start
Step2 : Read p,n,r
Step3 : calculate $I=(p \times n \times r)/100$.
Step4 : print I.
Step5 : stop.
```

Logic.

```
1. Start
2. p,n,r.
3. $I=(p*n*r)/100$.
4. Print I.
5. stop.
```

Selection

↳ Some portion of programs are executed based upon condition

↳ If condition is true, one part of the program will be execute, otherwise, other part of program will be executed.

Eg: Algorithm to find a number is even or odd.

Step1 : Start

Step2 : Read n

Step3 : If n%2==0 then goto Step 4,
 else goto step 5.

Step 4 : print 'Even'

Step 5 : print 'odd'

Step6 : End.

Logic.

```
1. Start
2. n
3. If n%2==0
4.      Even
   else
5.      odd
6. Stop
```

Iteration

Certain set of statements are executed again and again based upon condition.

Eg: Alg to print first 'n' natural numbers

Step1 : start

Step2 : Read n

Step3 : Initialize i as 1, count as 0.

Step4 : While count <n, then goto step 5,
 else goto step 9.

Step5 : print 'i'.

Step6 : Increment count by 1.

Step7 : Increment i by 1

Step8 : Goto Step 4.

Step9 : Stop.

Logic

```
1. Start.
2. n
3. i=1, count=0
4. count <n:
5.      print i.
6.      Count=count+1
7.      i=i+1
8. ↑
9. Stop.
```

Functions

↳ Subprograms that contains set of staliments that perform a particular task.

Benefits.

- ↳ Reduction in line of code.
- ↳ Code reuse
- ↳ Better readability.
- ↳ Improved maintainabilety.
- ↳ Easy to debug + test.

Eg: Alg to add 2 numbers

main()

```
Step1 : Start
Step2 : Call add()
Step3 : Stop
```

add()

```
Step1 : Read a,b.
Step2 : Compute
           C=a+b.
Step4 : print c
```

Logic

```
def add():       }
    a=5          }
    b=3          }  add()
    C=a+b        }
    print 'C'    }
add()
 ↑
main()
```

Simple strategies for developing algorithm

Two strategies for developing an algorithm

1. Iterations.
2. Recursions.

Iterations

↳ A sequence of statements are executed till specified condition is true.

1. For loop.
2. While loop.

For loop

↳ for iterating var in sequence:

```
for iterating var in sequence:
        Statements
```

↳

```
a = [1, 2, 3]
for i in a:
    print(i)
```

↳ o/p
1
2
3

While loop.

↳ initialization

```
initialization
While condition:
        statements
        increment/decrement
```

↳

```
i=1
While i<=3:
    print(i)
    i=i+1
```

↳ o/p
1
2
3.

Example: Sum of first 'N' Natural number.

Algorithm

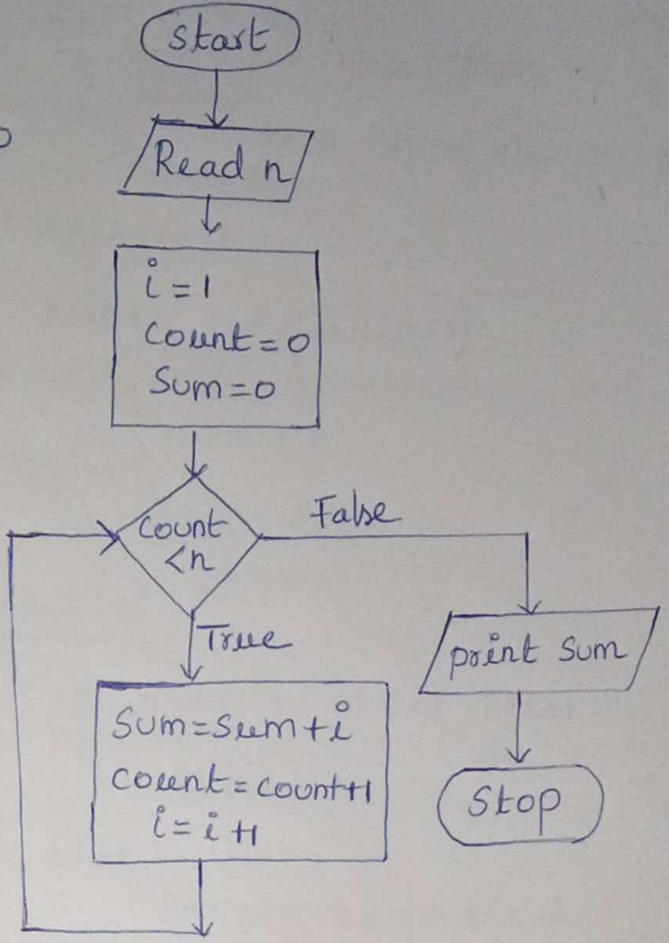
```
Step 1 : Start
Step 2 : Read n
Step 3 : Initialize i=1, count=0, Sum=0
Step 4 : While count<n, then goto step 5
             else goto step 9.
Step 5 : compute sum=sum+i
Step 6 : Increment count by 1
Step 7 : Increment i by 1
Step 8 : Goto step 4.
Step 9 : print Sum
Step 10 : Stop
```

1. start ▭
2. n ▱
3. i=1 , count=0 , Sum=0 ▭
4. count < n ◇
5. Sum = sum + i ▭
6. Count = count+1 ▭
7. i = i+1. ▭
8. ↑
9. print sum ▱
10. Stop. ▭

Pseudocode

```
BEGIN
GET n
INITIALIZE i=1, count=0, sum=0
WHILE count<n:
        COMPUTE SUM=SUM+i
        INCREMENT count by 1
        INCREMENT i by 1
END WHILE.
PRINT SUM
END.
```

flowchart



Recursion

↳ Recursion is a process by which a function calls itself repeatedly till some specified condition has been satisfied.

↳ A function that calls itself is known as recursive function.

Example: find factorial using recursive function.

```
1. Start ⬭
2. n ▱
3. z=fact(n) ⊟
4. print(z) ▱
5. Stop ⬭

fact(n): ⬭
1. if n==0 or n==1: ◇
2.        return 1
       else
3.        return
            n*fact(n-1)
```

Algorithm

main()

Step1 : Start

step 2 : Read n

Step 3 : Call fact(n) & Store it in z.

step 4 : print z

Step 5 : stop

fact(n)

Step 1 : if n==0 or n==1 then goto step2 else step3

Step 2 : return 1

Step 3 : return n*fact(n-1)

Pseudocode

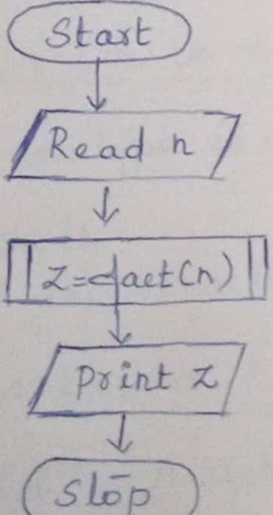
main()

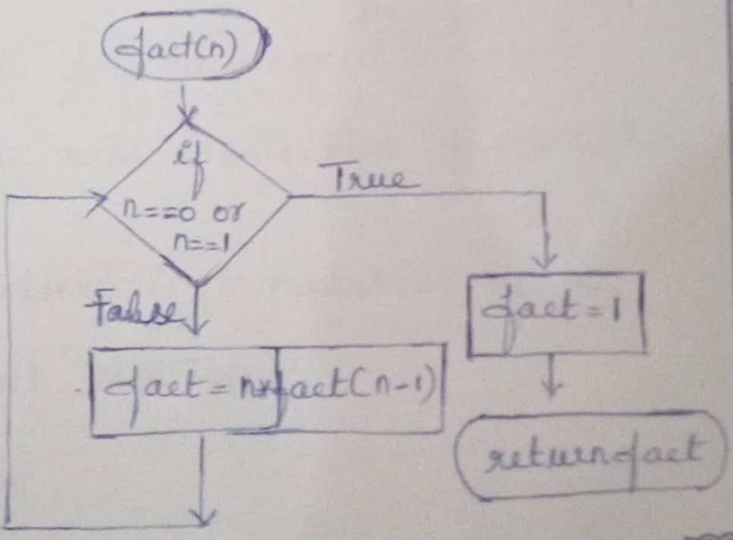
```
BEGIN
GET n
CALL fact(n) & Store it in z
PRINT z
END
```

fact(n)

```
IF n==0 or n==1 THEN
    RETURN 1
ELSE
    RETURN n*fact(n-1)
ENDIF
```

Flowchart





Algorithm Problem Solving

↳ Algorithm problem solving is solving the problem that require the formulation of an algorithm.

i) Understanding the problem

↳ Given problem should be completely understood.

↳ check whether it is similar to some standard problems or a known algorithm exists. otherwise, a new algorithm has to be devised.

ii) Ascertain the capabilities of computational device

↳ Done by knowing the type of architecture, speed and memory availability.

iii) Exact / approximate solution

↳ Exact solution problems.

a) Find addition of two numbers

b) Find given number is even or not

↳ Approximate solution pblm.

a) Find square root of a number.

b) Solving non-linear equation

iv) Decide on the appropriate data structure

↳ A datatype is a collection of data & collection of operations on it.

↳ A datastructure is an actual implementation of a particular ADT (Abstract Data Type)

Eg: Arrays, Queue, List

v) Algorithm Design Techniques

It is a general approach to solving the problems algorithmically that is applicable to a variety of problems.

vi) Methods of specifically an algorithm.

a) flowchart - Graphical representation of an algorithm

b) Pseudocode - Language representation of an algorithm.

vii) Proving an algorithm correctness

↳ The process of checking Whether an algorithm work correctly is called validation.

↳ It requires the solution be stated in two forms

a) set of assertions about input & output

viii) Analyzing an algorithm

↳ When an algorithm executed, it uses CPU to perform the operation & memory to Store the program & data.

↳ Analyzing an algorithm refers to the task of determining how much time & storage, an algorithm requires.

It gives quantitative judgements after the value of one algorithm over another.

Programming language

1) Interpreted programming language

It is a programming language whose implementations are typically interpreters.

Eg: Python, Ruby.

2) Compiled programming language

It is a programming language whose implementations are typically compilers.

Eg: C, C++

3) Functional programming language

↳ Specially designed to handle symbolic computation

↳ based on mathematical function.

Eg: Lisp, Haskell.

4) Procedural language

↳ Depends on predefined ~~&~~ well-organized procedures, functions or subroutines in program by specifying all steps that program must take to reach an output

↳ Procedures or functions are implemented to perform the task. It can be called anywhere in program & by other procedures as well.

Eg: C, Pascal

5) Object oriented programming lang

↳ Based on 'objects', Which contain 'data' & 'procedures'

Eg C++,

6) Scripting language

↳ Instructions are written for run-time environment.

↳ Not require compilation step & are rather interpreted

↳ Designed for integrating & communicating with other programming language.

Eg: Javascript, VB script.

7) Markup language.

↳ Designed for processing, definition & presentation of text.

↳ specifies code called tags for formatting.

Eg: HTML

8) concurrent programming language.

↳ provides technique for execution of operations Concurrently - either Within a single computer or across a number of systems.

Eg: Joule, limbo.

Electricity billing

Logic

1. Start
2. ~~Read U~~
3. ~~<=100~~
4. ~~U=U*0~~
5. ~~<=200~~
6.

Electricity Billing

Start

read U (U => Units)

<=100 free (U*0)

<=200 1-100 free

101-200 = U*1.5

<=500 1-100 free

201-200 => U*2

201 - 500 => U*3.

Logic

>500 1-100 free.

101-200 => U*3.5

201-500 => U*4.6

>500 => U*6.6.

print U

Stop.

Illustrative programs.

1) Find minimum element in list ✓

Algorithm.

Step 1: Start

Step 2: Read the list of numbers in a

Step 3: Assign first element as min

Step 4: For each element i in a, then
goto step 5 , else
goto step 7.

Step 5: if i<min, then goto step 6
else goto step 4

Step 6: set min=i then goto step 4

Step 7: print min

Step 8: Stop.

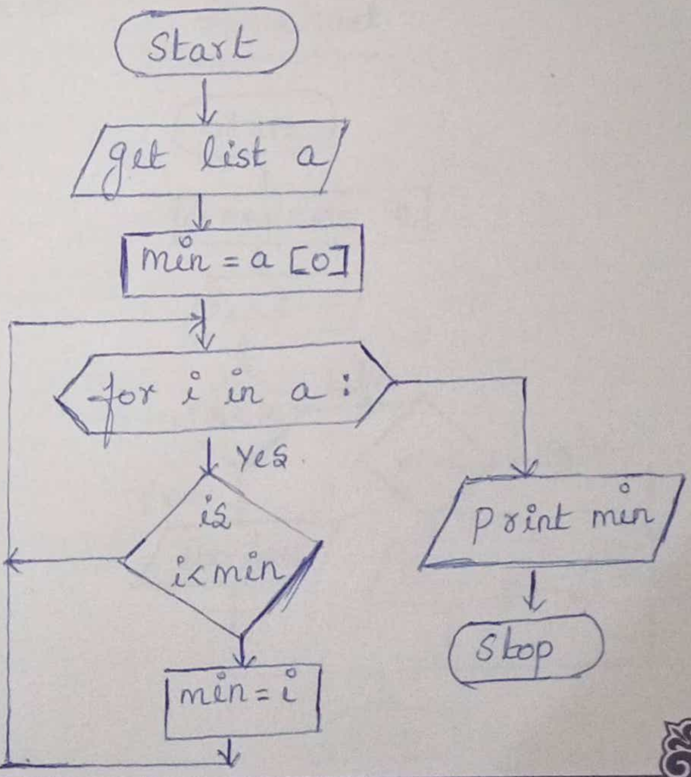
logic

```
1. Start
2. a=[10,20,30,40].
3. min=a[0].
4. for i in a:
5.     if i<min:
6.         min=i
7. print min
8. Stop.
```

Pseudocode

```
BEGIN
 READ list a
 SET min=a[0]
 FOR i in a:
     IF i<min:
         SET min=i
     ENDIF.
 ENDFOR
 PRINT min
END.
```

flowchart



2) Guess an integer number in a range ✓

Algorithm

Step 1: Start

Step 2: Generate random number between 1 and 100 in R

Step 3: Get guessing number from user in X

Step 4: if X<R, Then goto Step 5, else goto step 6.

Step 5: Print "Too low", then goto step 9

Step 6: if X>R, Then goto step 7, else goto step 8

Step 7: print "Too high", then goto step 9

Step 8: print "Guessing Correct"

Step 9: Stop.

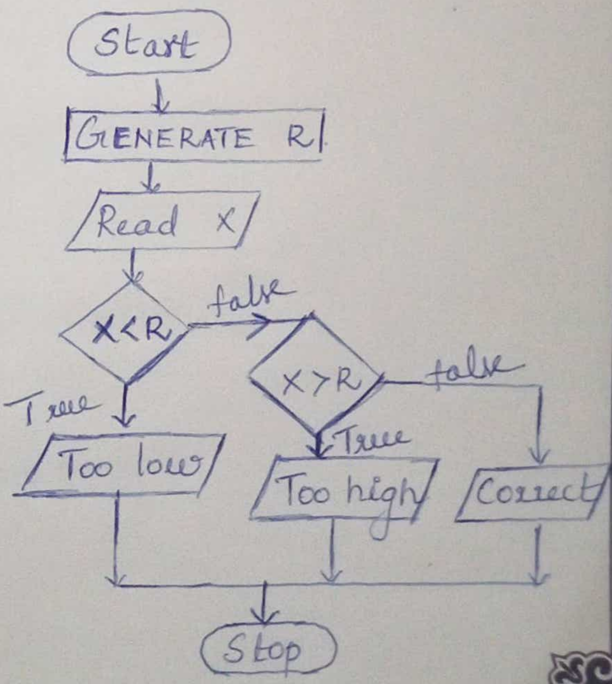
Logic

1. Start
2. R ⇐ random number (1 to 100)
3. X ⇐ guessing number
4. X<R
   Too low
5.
6. X>R
   Too high
7.
   else
8. Correct
9. Stop

Pseudocode

```
BEGIN
GENERATE random number R
GET guessing number X
IF X<R THEN
     PRINT "Too Low"
ELIF X>R THEN
     PRINT "Too high"
ELSE
     PRINT "GUESSING CORRECT"
ENDIF
END
```

flowchart



3) Insert a card into a list of sorted cards ✓

Algorithm

Step1: Start

Step2: Read list of sorted cards.

Step3: First card is already sorted

Step4: Pick next card.

Step5: Compare with all cards in the sorted Sub-list.

Step 6: Shift all cards in sorted sublist that is greater than value of card. to be inserted.

Step 7: Insert a card

Step8: Repeat from step4 till all cards are sorted.

Step 9: Stop

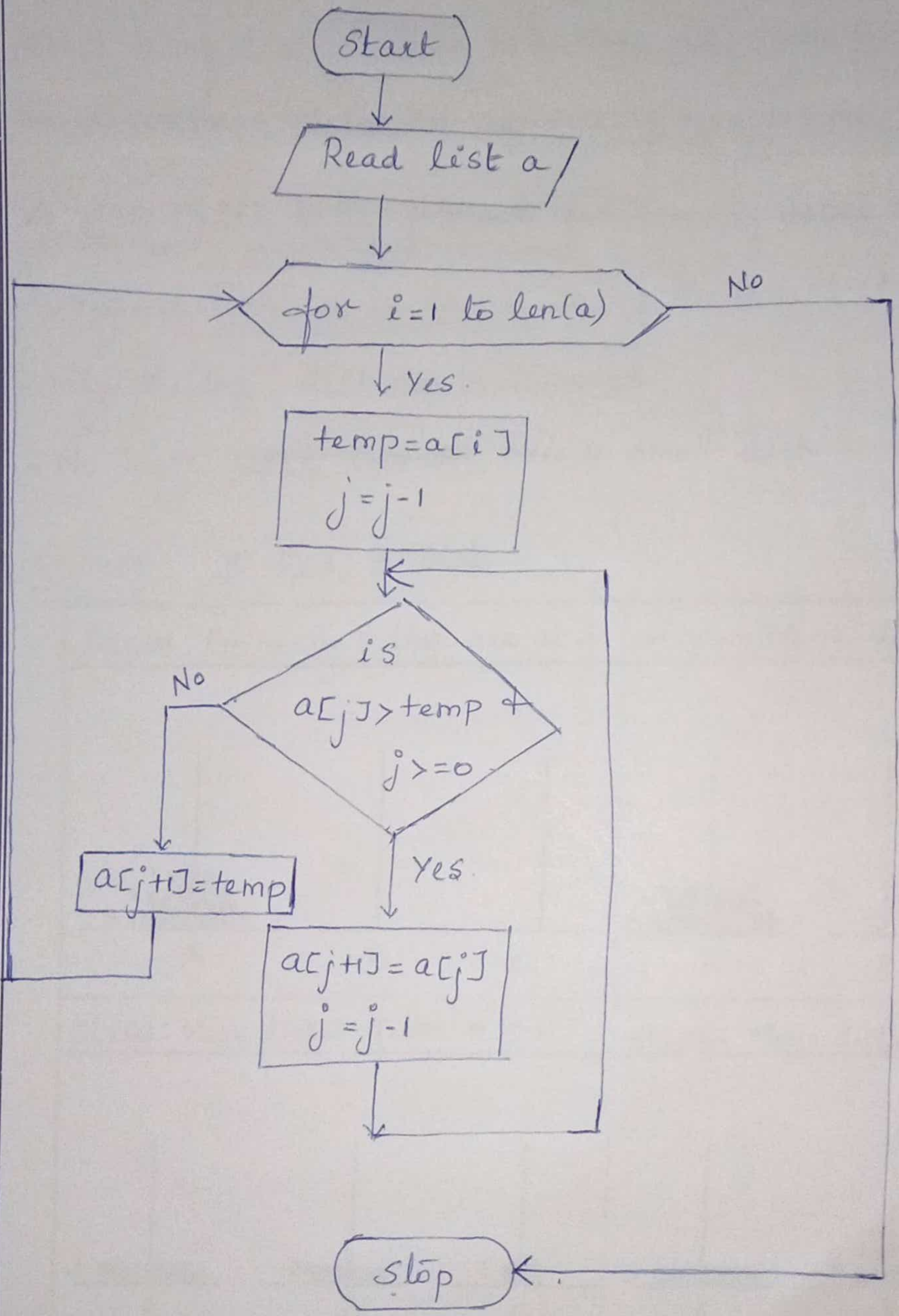
Pseudocode

```
BEGIN
 READ list a
 FOR  i=1 to len(a):
        temp = a[i]
        j=1
        WHILE j>=0 and j a[j] >temp:
                      a[j+1] = a[j]
                      j = j-1
        ENDWHILE
        a[j+1] = temp
END FOR
PRINT  a
```

Logic

```
Start
Read list a
for i=1 to len(a):
    temp=a[i]
    j=i-1
    While j>=0 and
           a[j]>temp:
      a[j+1]=a[j]
      j=j-1
   a[j+1]=temp
print a
Stop.
```

Flowchart


Start
Read list a
for i=1 to len(a)
No
Yes.
temp=a[i]
j=j-1
is
a[j]>temp &
j>=0
No
Yes.
a[j+1]=temp
a[j+1] = a[j]
j = j-1
stop

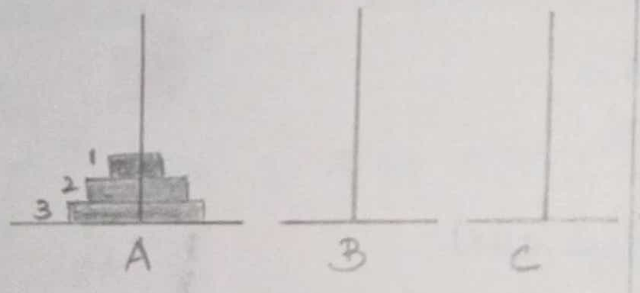
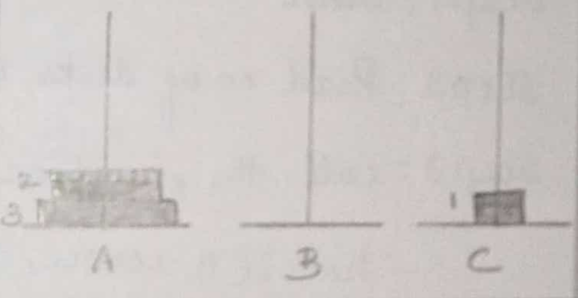
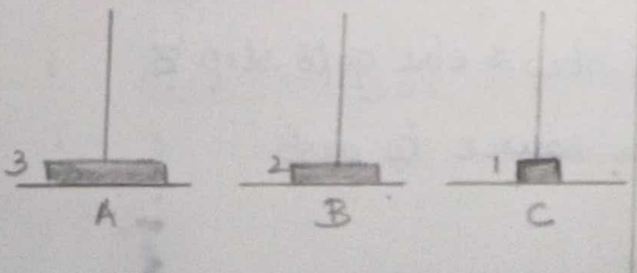
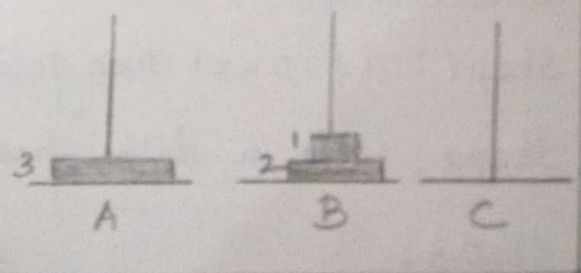
Towers of Hanoi ✓

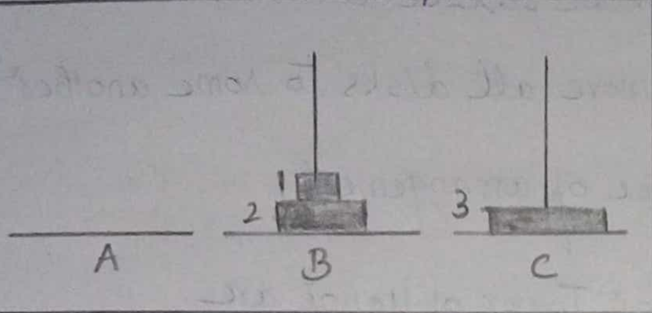
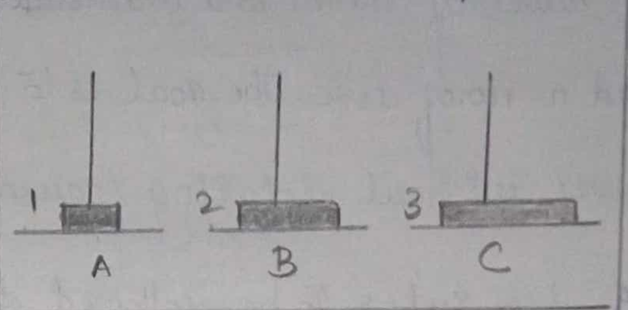
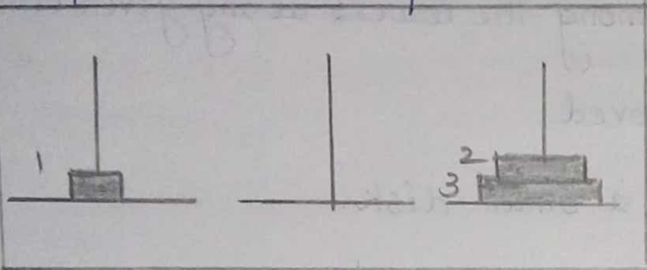
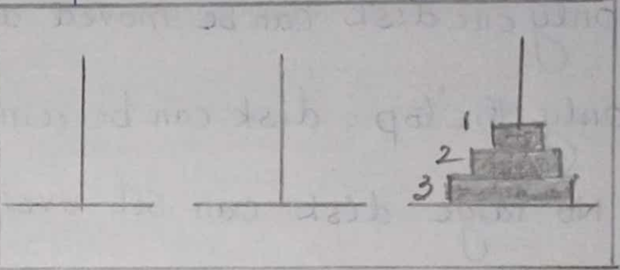
↳ Tower of hanoi is a mathematical puzzle with three rods and n no.of disc. The goal is to move all disks to some another tower without violating sequence of arrangement.

A few rules to be followed for Tower of Hanoi are

- only one disk can be moved among the towers at any given time
- only the 'top' disk can be removed
- No large disk can sit over a small disk.

Given 3 Rod, 3 Disk

Step1: Initially 3 disk are at A	Step2: Move disk 1 from A to C
	
Step3: Move disk 2 from A to B	Step4: Move disk 1 from C to B
	

Step5: Move disc3 from A to C	Step6: Move disk1 from B to A
	
Step7: Move disk2 from B to C	Step 8: Move disc 3 from A to C
	

Algorithm.

```
main()
Step1: Start
Step2: Read no.of disks n
Step3: Call the function
        hanoi(n, source, dest, aux)
Step4: Stop

hanoi(n, source, dest, aux)
Step1: if n==1 then goto step 2 else goto step 3
Step2: Move n disk from source to dest
Step3:
```

```
hanoi( n-1, source, aux, dest)
move nth disk from source to dest
hanoi(n-1, aux, dest, source)
```

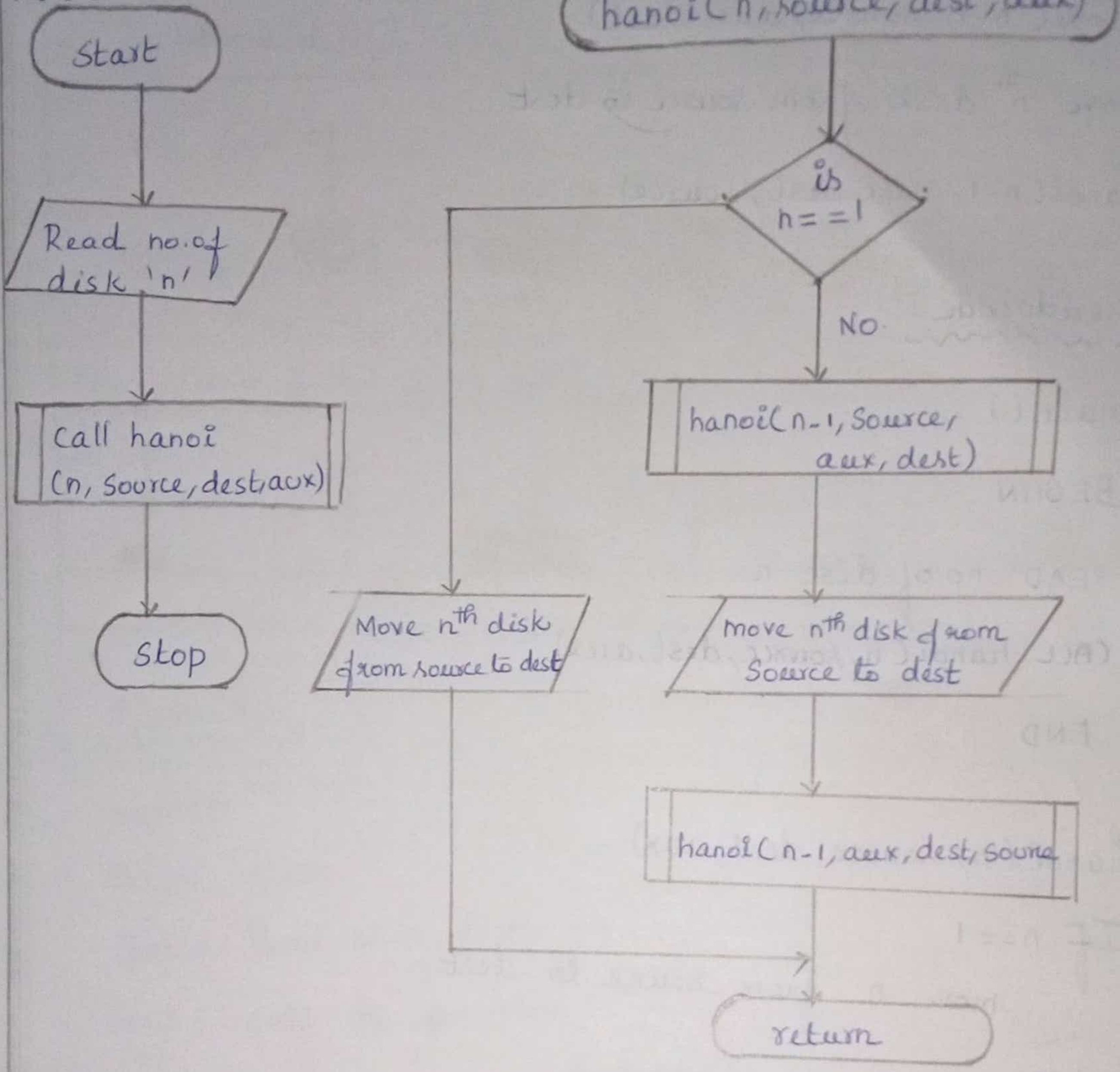
pseudocode

```
main()
BEGIN
 READ no.of disc n
 CALL hanoi(n, source, dest, aux)
 END

hanoi (n, source, dest, aux)
If n==1
     move n from source to dest
ELSE
     hanoi(n-1, source, aux, dest)
     move nth disk from source to dest
     hanoi (n-1, aux, dest, source)

ENDIF
```

flowchart


Start
Read no.of disk 'n'
Call hanoi (n, source, dest,aux)
Stop
hanoi(n, source, dest, aux)
is n==1
No
hanoi(n-1, Source, aux, dest)
Move nth disk from source to dest
move nth disk from Source to dest
hanoi(n-1, aux, dest, source
return

UNIT-2.

Python interpreter

↳ python is an interpreted programming language, Whose implementations are interpreters (Step-by-step executors of sourcecode, Where no pre-runtime translation takesplace (ie) source code is translated to machine code Step-by-Step While the program is being executed)

Modes of python interpreter.

1. Interactive mode.
2. Script mode

Interactive mode.

↳ Interact with OS.

↳ When the python expression/statement/command typed after >>>, the python immediately responses with output of it.

Eg:

```
>>> print('Hai')
Hai
>>> print(10 *2)
20
```

Pros:

1. Easy to run only a single or few lines of code

2. Get immediate results.

3. Good for beginners Who need to understand phython basics.

Cons:

1. Editing the code is hard

2. Tedious to run long piece of code.

Script mode

The program can be stored in a file & interpreter can be used to execute the file.

Note: filename ends with .py extension

Run as Run -> Run module

Eg.

add.py

```
a = int(input('Enter a value:'))
b=int(input('Enter b value:'))
C=a+b
print(c)
```

Output

```
Enter a value:5
Enter b value :10
15
```

pros.

1) Easy to run long piece of code

2) Editing the code is easy

3) Good for both beginners & experts

Cons

1) Tedious to run only a single or few lines of code.

2) Must create & save file before executing code

2. Values & Types

Value: ↳ Value is any number / letter / string

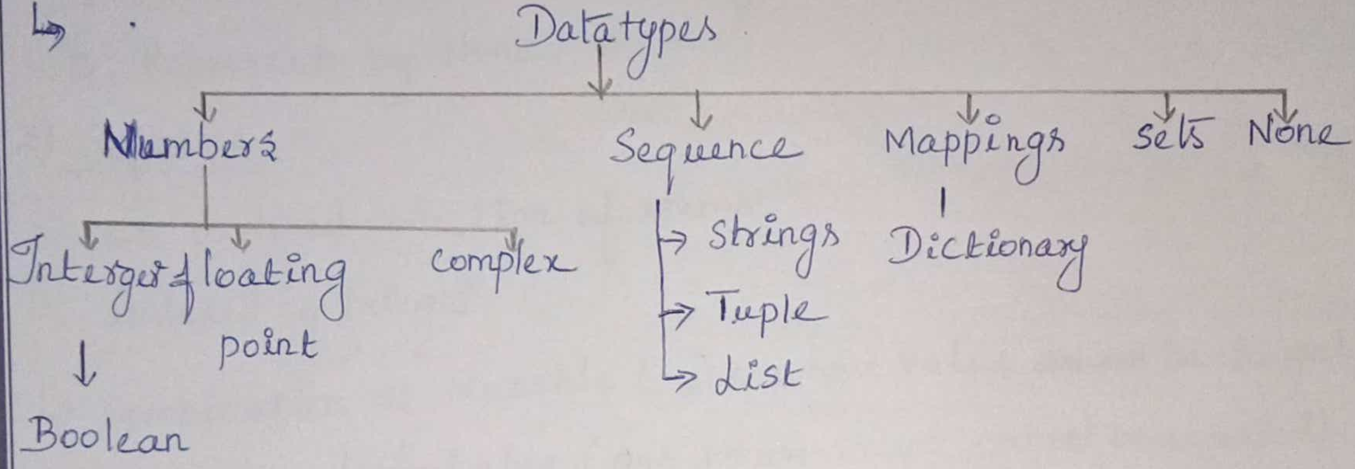
Eg: 10, 'a', 'Hello World!'

↳ To assign value to a variable, use assignment operator (=)

Eg: a = 10

Type (ie datatype): ↳ Datatype is a set of values & allowable operations on those values.

↳



1. Number

↳ stores Numerical value

↳ Immutable - value cannot be changed.

Types

Integer - To store whole numbers without fraction part.

Eg: 10, -20, 1234L.

Float - To store number with fraction part

Eg: 3.45

Complex : To store real/imaginary part.

Eg: 8+9j.

Boolean : Integers contain boolean type, consisting of two constants, True or False.

Eg : flag = True

2) None

↳ This is special datatype with single value,

↳ Used to signify the absence of value in a situation.

↳ Represent by None.

3) Sequence

↳ an ordered collection of items.

↳ indexed by integer.

↳ Combination of Mutable (one whose value ~~cannot~~ be changed)

Immutable (one whose value cannot be changed)

Types

String : ↳ ~~sequa~~ collection of letter / character.

↳ Enclosed in ' ' or " "

↳ String with length 1 represents character.

↳ Immutable

↳ Indexed by positive or negative integer

Eg a = 'Student'

Operations
1) Creation ‘ ’
2) Indexing [ ]
3) Slicing [:]
4) Repetitions *
5) Concatenation +

List :

- Collection of values of any type.
- Enclosed in square bracket [ ]
- Mutable
- Indexed by positive or negative integer.
- Values also called as elements / items.

Eg.
a = [1, 'xyz']

Operations:
1) Creation [ ]
2) Indexing [ ]
3) Slicing [:]
4) Repetitions *
5) Concatenation +

Tuple

- Collection of values of any type
- Enclosed in parenthesis ( )
- Immutable
- Indexed by positive or negative integer
- values in tuple are called as elements / items.

Operations
1) Creation ( )
2) Indexing [ ]
3) Slicing [:]

4) Repetitions *

5) Concatenation +

4) Sets

- Unordered collection of values of any type.
- No duplicate entry.
- Immutable.

Eg S = {1, 20, 'xyz'}

5) Mapping

Dictionary

- Unordered collection of values of any type.
- Enclosed in curly braces { }.
- Mutable.
- indexed by key.
- Store in key-value pair.

Eg: d= {1: 'a', 2: 'b', 3: 'c'}.

Identifiers

- defined as names given to entities like class, function, variables etc
- combination of letters in lowercase (a to z), Uppercase (A to Z), digits (0 to 9) or an underscore (_)
- Not start with digit
- keywords & special symbols like !, @, #, $, % etc cannot be used.
- can be of any length.

Variables

↳ defined as an identifier that refer to a value.

While creating a variable, memory space is reserved in memory.

Based on the datatype of a variable, interpreter allocates memory

↳ values can be assign to variables using assignment operator(=)

Eg: a = 'Student' // a is a variable

↳ combination of a to z, A to Z, 0 to 9, underscore(_)

↳ Not start with digit

↳ keywords & special symbols like !, @, #, $, % etc cannot be used.

↳ can be of any length.

Keywords

↳ Reserved words

↳ cannot use keyword as identifier

↳ define the syntax & structure of python lang

↳ case sensitive.

Eg

and
or
for
While

if
elif
else
print

continue
break
is
in

try
except
finally
raise

def
return
True
False

Expressions & statements

↳ Expression defined as combination of value, variable & operator.

↳ A value, a variable itself an expression.

Eg: x, 17, x+17.

↳ A Statement is a code that ends with a newline character.

↳ Use line continuation character (\) to denote that the line continue.

↳ cannot be used within list, tuple or dictionary.

Eg

```
Total = mark1 + \
        mark2 + \
        mark3
```

↳ 2 kinds of statement

Print statement

Assignment statement

Eg

```
>>> Str = 'Hai'
>>> print(Str)
>>> Hai.
```

Comments

↳ Non executable statements that explain What program does.

↳ added in program with symbol #

↳ Helpful for Large programs, since it difficult to understand

Program

```
# program to create list
a = []
n = int(input('Enter no.of elements:'))
for i in range (0, n):
    z = int(input('Enter element'))
    a.append(z)
```

Tuple Assignment

↳ Useful to swap the values of 2 variables

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

$$\left.\begin{array}{l}\text{left side is tuple of variables}\\ \text{Right side is tuple of values}\end{array}\right\}\begin{array}{l}\text{Each value is assigned to its}\\ \text{respective variables.}\end{array}$$

↳ Right side is evaluated before any of the assignments.

↳ It can be any kind of sequence.

↳ No. of variables on left & No. of values on right have to be same.

```
>>> a, b = 1, 2, 3
ValueError: Too many values to unpack.
```

Usage of split function.

```
>>> addr = 'be@gmail.com'
>>> uname, domain = addr.split('@')
```

The return value from split is a list with 2 elements.

The first element is assigned to uname, the second one to domain

```
>>> uname        >>> domain
   'be'           'gmail.com'
```

Operators

An operator is a symbol that performs operation on operand.

Eg: >>> $\underbrace{4 \overset{\text{operator.}}{+} 5}_{\text{operands}}$

Types.

1. Arithmetic operators

Perform arithmetic operations on operands.

operators: +, -, *, /, //, %, **

Eg:

```
a = 10
b = 2                    o/p
print(a+b)               12
print(a-b)               8
print(a*b)               20
print(a/b)               5.0
print(a//b)              5
print(a%b)               0
print(a**b)              100.
```

2) Relational operators

Compare the value of operands & return either true or false based on condition

operators: ==, !=, >, <, >=, <=

Eg:

```
a=10
b=2                       O/P
print(a==b)               False
print(a!=b)               True
print(a>b)                True
print(a<b)                False
print(a>=b)               True
print(a<=b)               False.
```

3) Assignment operator.

Used to assign value to operand.

Operator : +=, -=, *=, /=, //=, %=, **=

Eg:
```
a=10
b=2                  O/P.
print(a+=b)          12
print(a-=b)          8
print(a*=b)          20
print(a/=b)          5.0
print(a//=b)         5
print(a%=b)          0
print(a**=b)         100.
```

4) Logical operator.

Compare the values of operands, & return either True or False based on condition.

Operators: and, or, not

Eg:
```
                                  O/P:
a=10
b=2
print( 10!=2 and 10>2)            True
print(10!=2 or 10>2)              True
print(not 2)                      False.
```

5) Bitwise operator.

perform bit by bit operations on operands.

operators: &, |, ^, ~, <<, >>

Eg

```
a=10.
b=2
print(a&b)
print(a|b)
print(a^b)
print(~a)
print(a<<b)
print(a>>b)
```

5) ~~Bitwise operators~~ 6) Membership operator

~~perform~~

Check for membership in sequence & return either True or False based on condition.

operators: in, not in.

Eg

```
a=[10,20,30,40]        o/p
b=20
print(b in a)          True
print(b not in a)      False.
```

7) Identify operator.

↳ Compare the values of operands & return either True or False based on condition.

Operators: is, is not

Eg:

```
a=10                   o/p
b=100
print(a is b)          False
print(a is not b)      True
```

8. Unary arithmetic operator

Returns its numeric argument with or without sign

operators: +, -

```
Eg:  a = 10                 o/p
     print(+a)              10
     print(-a)              -10
```

Operator precedence

precedence.

When more than one operator appears in an execution, the order of execution depends on the rules of precedence

Associativity

If there are two operators in an expression with same level of precedence, then associativity

	Operator	Description	Associativity
1.	( )	paranthesis	Left to Right
2.	**	Exponential	Right to left
3.	+	unary plus	Left to Right
	-	Unary Minus	
	~	Bitwise one complement	
4.	*	Multiplication	left to Right
	/	Division	
	//	Floor division	
	%	Modules	

5. + −	Addition subtraction	Left to Right
6. << >>	Left shift Right shift	Left to Right
7. &	Bitwise AND	Left to Right
8. ^	Bitwise XOR	Left to Right
9. |	Bitwise OR	Left to Right
10. in, not in is, !=, ==, <, <=, >, >=,	Membership Identity Relational	Left to Right
11. +=, -=, *=, /=, //=, %=	Arithmetic	$\boxed{\text{Right to left}}$
12. not 13 and 14. or	logical not logical and logical or	Left to Right

Eg: 2 + 3 * 4 // 5 - 17 solve it.

12

2

4

$\boxed{-13}$

Functions.

A function is a group of statements that perform a specific task It contains line of codes that are executed sequentially from top to bottom by python Interpeter.

It can be categorized into

i) Modules
ii) Built-in
iii) User defined.

Built-in function

↳ Functions that are already built into python interpreter and are readily availyable for use.

Name	Describe	Example
1. abs(x)	It returns distance b/w X and zero X → numeric value	>>> abs(-45)
2. max(x,y,z,...)	Returns largest of its arguments x,y,z → numeric value	>>> max(80,100) 100.
3. min(x,y,z,...)	Returns smallest of its arguments x,y,z → numeric value	>>> min(80,100) 80.
4. cmp(x,y)	Returns the sign of the difference of two numbers	>>> cmp(80,80) 0

	-1 if x<y
	0 if x==y
	1 if x>y
	x,y→ numeric value
		>>> cmp(80,80)
		0
		>>> cmp(80,100)
		-1
		>>> cmp(80,40)
		1

5) divmod(x,y)	Returns both quotient & reminder by division ↳ x is divided by y ↳ x,y → numeric value	>>> divmod(14,5) (2,4)

6) len(s)	Return the length of its arguments.
	S ⇒ sequence/mapping
		>>> a = 'Hello'
		>>> len(a)
		5

7) range([start,], stop, [, step])	↳ Versatile function to create list containing arithmetic progressions ↳ start, stop, step ⇒ integer X· If start is omitted, it defaults to 0 If step is omitted, It defaults to 1	>>> range(4) [0, 1, 2, 3]
8) round(x[, n])	↳ Returns x rounded to n digits from decimal point ↳ If n is omitted, x is rounded to 0 decimal point	>>> round(12.345,2) 12.35

8) ~~round(~~ some more functions are

int()   str()

long()   list()

float()   tuple()

bool()   chr()

User defined functions

The functions defined by user according to their requirements are called user defined functions.

↳ Defining a function

↳ Calling a function

↳ Defining a function

Defining a function

↳ Using the keyword def followed by function name & parenthesis.

↳ Includes.

* Header, begins with def and end with colon (:)
* Body, consists of one or more python statements

Syntax

```
def function_name(parameters):
        [function_docstring]
        function_statements
        return [expression]
```

Where,

function_docstring → documentation of the function Which is optimal.

return [expression] → return result of the function & exit.

Calling a function

↳ Functions can be executed when it is called.

↳ can be called from another function or directly from the prompt by its name.

Syntax:

```
function_name(parameters)
```

Eg:

```
def display():
    print('Welcome')

display()
```

Flow of execution

↳ Specifies the order in Which statements are executed.

↳ program execution starts from first statement

↳ one statement executed at a time.

↳ Function definitions do not alter the flow of execution of programs.

↳ Function statements are executed only When it is called

↳ When a function is called, control flow jumps to the body of function, executé & return back to place Where it was called.

↳ When reach the end of program, it terminates.

Parameters & Arguments.

↳ Arguements are variables / values passed through function call.

↳ Parameters are variables used in function definition to get values passed as an arguments.

Types

1. Function without arguement & without return type.
2. Function without arguement & with return type
3. Function with argument & without return type
4. Function with argument & with return type.

Eg

1. W/o arguments, W/o ~~With~~ return

```
def add()
    a=int(input('Enter a value:'))
    b=int(input('Enter b value:'))

    c=a+b
    print(c)
add()
```

o/p

```
Enter a value: 5
Enter b value: 3
8
```

2. W/o argument, with return

```
def add():
    a=int(input('Enter a value:'))
    b=int(input('Enter b value:'))
    c=a+b
    return c
z=add()
print(z)
```

O/p

```
Enter a value:5
Enter b value:3
8
```

3) With argument, w/o return

```
def add(a,b):
    c=a+b
    print(c)

a=int(input('Enter a value:'))
b=int(input('Enter b value:'))
add(a,b)
```

O/p

```
Enter a value:5
Enter b value: 3
8
```

4) With argument, with return

```
def add(a,b)
    c=a+b
    return c

a=int(input('Enter a value:'))
b=int(input('Enter b value:'))
z=add(a,b)
print(z)
```

O/p

```
Enter a value:5
Enter b value:3

8
```

Module.

↳ Module is a file that contains a collection of related functions. To use these modules, programmer needs to import the module in program.

↳ 4 ways to import a module.

1) Import:

Eg
```
import math
x=math.sqrt(25)
print(x)
```

o/p// 5.0.

2) from import:

Eg
```
from math import sqrt
x=sqrt(25)
print(x)
```

o/p// 5.0.

3) import with renaming:

Eg
```
import math as m
x=m.sqrt(25)
print (x)
```

o/p// 5.0.

4) import all

Eg
```
from math import *
x = sqrt (25)
print(x)
```

o/p// 5.0

Types of Modules

1) Built-in modules.

2) User defined modules

Built in modules.

↳ Math, random are built in modules that are available in python to support familiar mathematical functions.

↳ import it to use in program.

Some function in math module :

Function	Description	Example.
1. floor(n)	Round down to nearest integer	>>>math.floor(4.7) 4.0
2. ceil(n)	Round up to nearest integer	>>> math.ceil(4.7) 5.0
3. pow(n,d)	Return n raised to power d.	>>>math.pow(10,3) 1000
4. sqrt(n)	Return squareroot of 'n'	>>> math.sqrt (16) 4
5. factorial(n)	Return factorial of 'n'	>>>math.factorial(5) 120
6. gcd(n,m)	Return gcd of n,m	>>>math.gcd(10,2) 2
7. trunc(x)	Return truncated value of x	>>>math.trunc(1.99) 1

8. Sin(x)
cos(x)
tan(x)
	Return sine, cosine, tangent of x	>>> math.sin(math.pi/4)
		0.70.

Some functions in random module are:

Function	Description	Example
1) random()	Return float x 0 ≤ x < 1	>>> random.random() 0.25
2) randint(a,b)	Return int x a ≤ x < b	>>> random.randint(1,10) 5
3) uniform(a,b)	Return float x a ≤ x < b	>>> random.uniform(5,10) 5.5
4) randrange([start,]stop[,step])	Return x from given range	>>> random.randrange(100,100,3) 150.

User defined modules

python allows to define ourown modules.

Eg

Sample.py

```
def add(x,y):
    z=x+y
    return z
```

Run as.

```
>>> import Sample ~~module~~
>>> x = Sample.add(5,4)
>>> print(x)
9
```

Arguments types

1) Required Arguments.

The no.of arguments in function call match exactly with function definition.

Eg

```
def my_details(name,age):
        print(name,age)

my_details('Rose', 20)
```

O/P

```
Rose 20
```

2) keyword Arguments

Python interpreter use the keywords provided in arguments to match it with parameters even though if they are arranged in out of order.

Eg

```
def my_details(name, age):
        print(name, age)
my_details(age = 20, name='Rose')
```

O/P

```
Rose 20
```

3) Default Arguments.

Python interpreter use the default values when it is not provided in function call.

Eg

```
def my_details(name, age=40):
        print(name, age)

my_details('Rose')
```

O/P

```
Rose 40
```

4) Variable length Arguments.

↳ Used to specify more parameters while defining function

↳ denoted by * symbol before parameter.

Eg

```
def my_details(*name):
        print(*name)
    my_details('Rose', 'Roshan', 'Rojar')
```

o/p//

```
Rose Roshan Rojar
```

Structure of python program

↳ Python line structure.

- physical line is sequence of characters.
- logical line contains only spaces, & tabs which are ignored by python interpreter.

Eg:

```
x=1
if x>0:
    print ('Hello')
```

↳ comments.

- A comment begins with #
- All characters after # are ignored by python interpreter.

Eg:

```
# program to print Hello
x=1
if x>0:
    print('Hello')
```

↳ Joining two lines:

Backslash character (\) is used to join 2 lines.

Eg:

```
x=1
x=2
if x==1 \
        and x==2:
            print('Hello')
```

↳ Multiple statements on a single line

Semicolon character(;) are used to write multiple statements on single line

Eg

```
print ("Hai") ; print("Hello")
```

↳ Identation

Identation are used to define program block

All statements within block must be intended same amount

Eg

```
x=1
if x>0:
        print('Hai')
        print('Hello')

print('Welcome')
```

O/P

```
Hai
Hello
welcome
```

UNIT-III

1. Conditionals

Boolean values and operators

↳ The Boolean values are True & False.

↳ The Boolean operators are relational & logical operators

↳ Boolean expression - statements that prints either true or false.

Relational operator.

↳ compare the values of operands and return either True or False based on condition.

Operators : <, <=, >, >=, ==, !=

```
program.                    Output

  a = 10
  b = 2
  print (a<b)               False
  print (a<=b)              False
  print (a>b)               True
  print (a>=b)              True
  print (a==b)              False
  print (a!=b)              True
```

logical operators

↳ Compare the values of operands and return either True or False based on condition.

Operators : and, or, not

Program.

```
a=10                              Output
b=2
print (a>b and a<b)               False
print (a>b or a<b)                True
print (not a)                     False
```

2) Conditional Execution

Types.

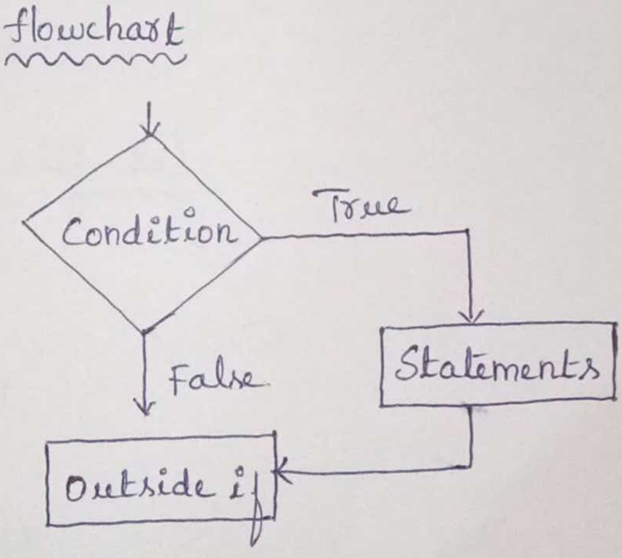
1. Conditional if
2. Alternative if...else
3. Chained if...elif...else
4. Nested if...else.

Conditional (if)

Syntax

```
if condition:
       statement
```

flowchart



Program.

```
x=5
 if x==5:
   print('Hai')
```

O/P

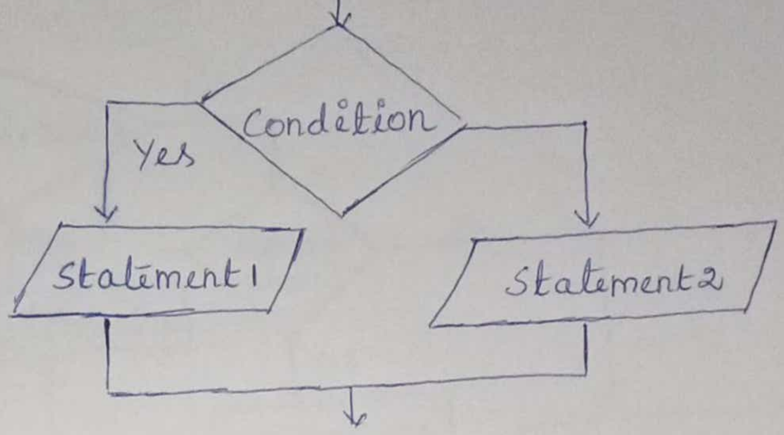
```
Hai.
```

Alternative (if...else)

Syntax.

```
if condition:
        Statement 1
else:
        Statement 2
```

flowchart



Program:

Greatest among 2 Nos

```
x = int(input('Enter x value:'))
y=int(input('Enter y value:'))
if x>y:
      print('x is greatest')
else:
      print('y is greatest')
```

O/P

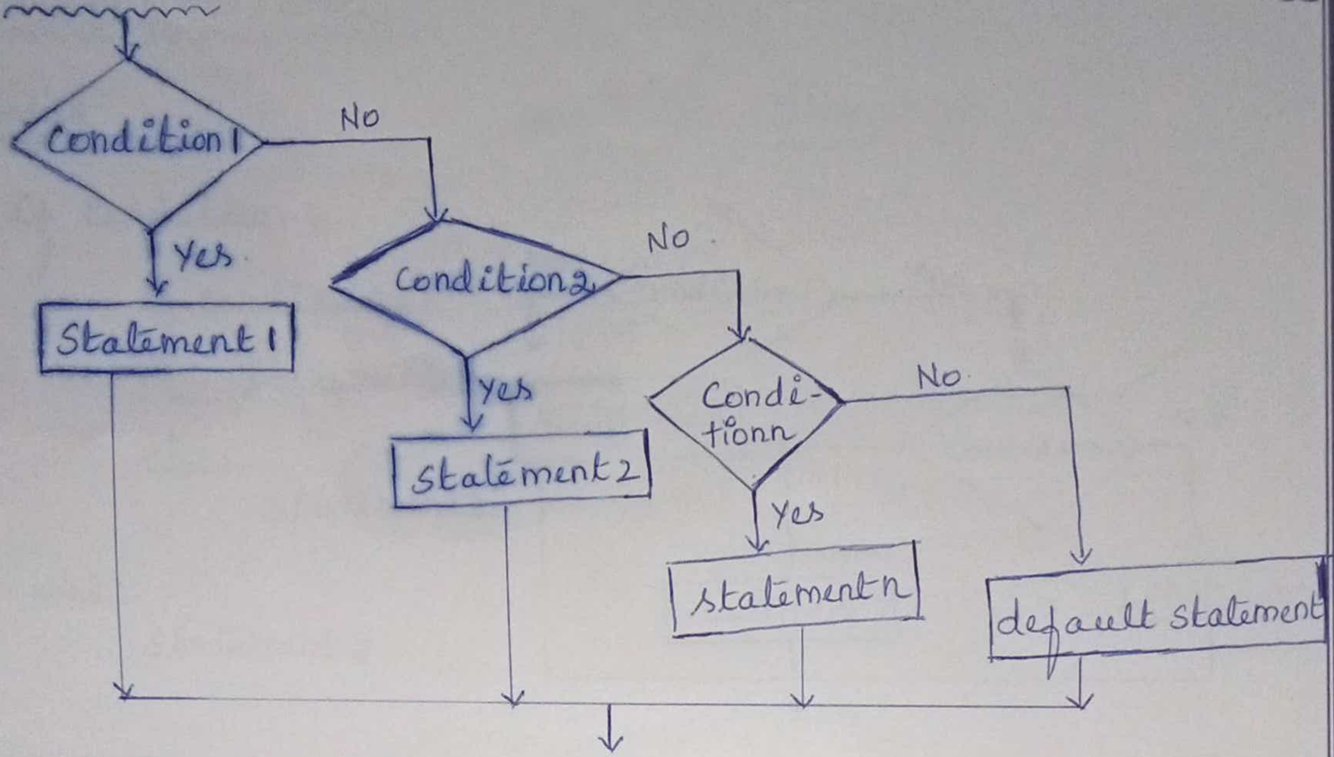
```
Enter x value : 10
Enter y value : 20.
y is greatest.
```

Chained conditionals (if-elif-else)

Syntax.

```
if condition 1:
        statement1
elif condition2:
        statement2
 :
 :
else:
      default statement
```

flowchart



Program.

```
a = int(input('Enter a value:'))
b= int(input('Enter b value:'))
c=int(input('Enter c value:'))
if  a>b and a>c:
     print('a is greatest')
elif  b>c:
      print('b is greatest')
else:
      print('c is greatest')
```

O/p

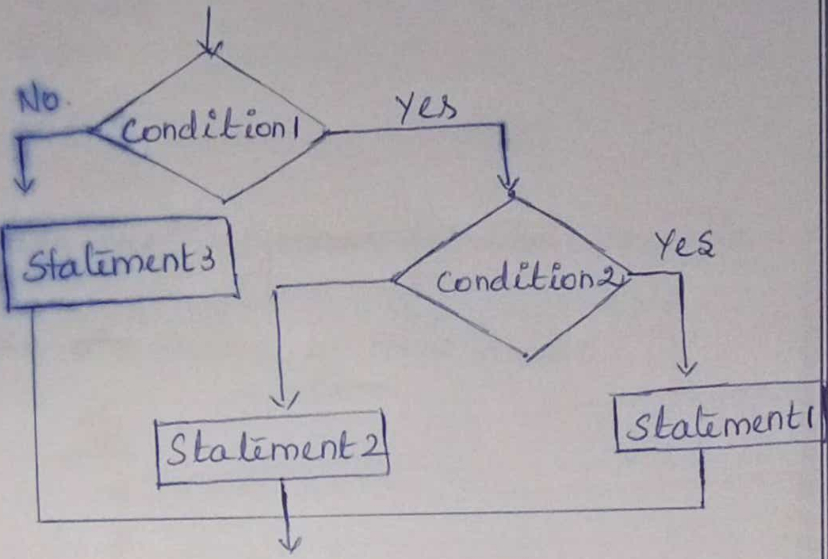
```
Enter a value: 10
Enter b value : 20
Enter c value :15
 c is greatest
```

Nested conditionals

Syntax

```
if condition1:
    if condition2:
        Statement1
    else:
        Statement2
else:
    statement3
```

flowchart



Program

```
a=int(input('Enter a value:'))
b=int(input('Enter b value:'))
c=int(input('Enter c value:'))
if a>b:
    if a>c:
        print('a is greatest')
    else:
        print('c is greatest')
elif b>c:
    print('b is greatest')
else:
    print('c is greatest')
```

Output

```
Enter a value:10
Enter b value:20
Enter c value:15
b is greatest.
```

3. Iteration statements

- State
- While
- For.

State:

↳ Possible to have more than one assignment for same variable.

↳ New assignment replace old value by New value.

Eg

```
x=5                    o/p
y=3                     4
x=4                     3.
print(x,y)
```

While.

↳ Condition is checked first, if it true, the body of the loop is entered.

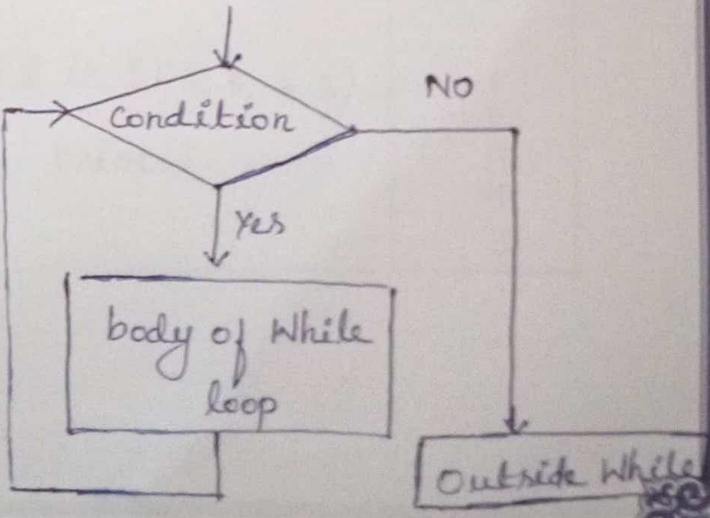
↳ After one iteration, condition is checked again.

↳ This process continues until condition evaluates to False.

Syntax.

```
initial value
While condition :
      body of While loop
      increment /Decrement
```

Flowchart.



For loops.

For in sequence

↳ For loop is used to iterate over sequence (list, tuple, string). Iterating over a sequence is called traversal. Loop continues until we reach the last element in the sequence

↳ The body of for loop is seperated from rest of the code using indentation.

Syntax

```
for i in sequence:
    print(i)
```

S.No	Sequence	Example	Output

1.	For loop in string	for i in 'Raja':
		print(i)
			R
			a
			j
			a

2.	For loop in list	for i in [1,2,3,4,5]:
		print(i)
			1
			2
			3
			4
			5

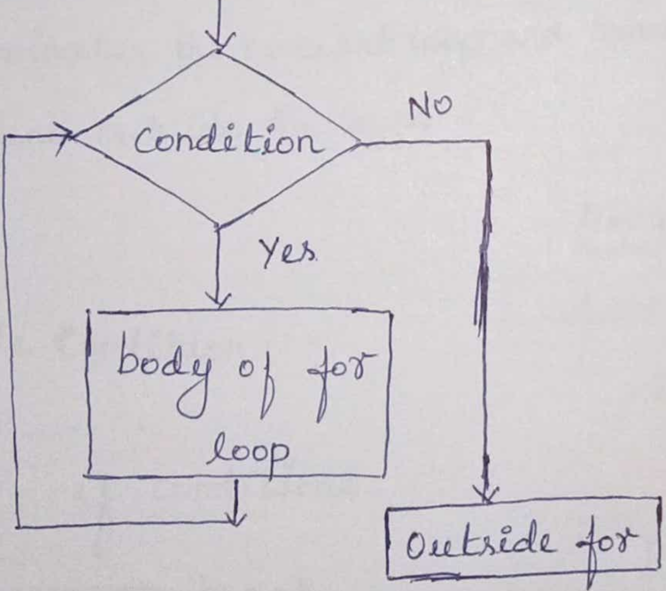
3.	For loop in tuple	for i in (1,2,3,4,5):
		print(i)
			1
			2
			3
			4
			5.

For in range.

To generate sequence of numbers range() is used.

1. range(n) # generates number from 0 to n-1	for i in range(5): print(i)	0 1 2 3 4
2. range(m,n) # generates numbers from m to n-1	for i in range(3,6): print(i)	3 4 5
3. range(m,n,x) # generates number from m to n-1 with skip x	for i in range(3,8,2): print(i)	3 5 7

flowchart



Program

```
a=0
b=1
n= int(input('Enter no.of terms:'))
print (a,b)
for i in range(1,n+1, 1):
        c=a+b
        print(c)
        a=b
        b=c
```

Output

```
Enter no.of terms:5

0 1
1
2
3
5
8
```

4. Loop control structure

Loop control statements are used to change the program execution from its normal sequence.

Break

It terminates the current loop and transfer the control to statement outside the loop.

Syntax

```
While Condition1 :
        ....
        if condition2:
                break
```

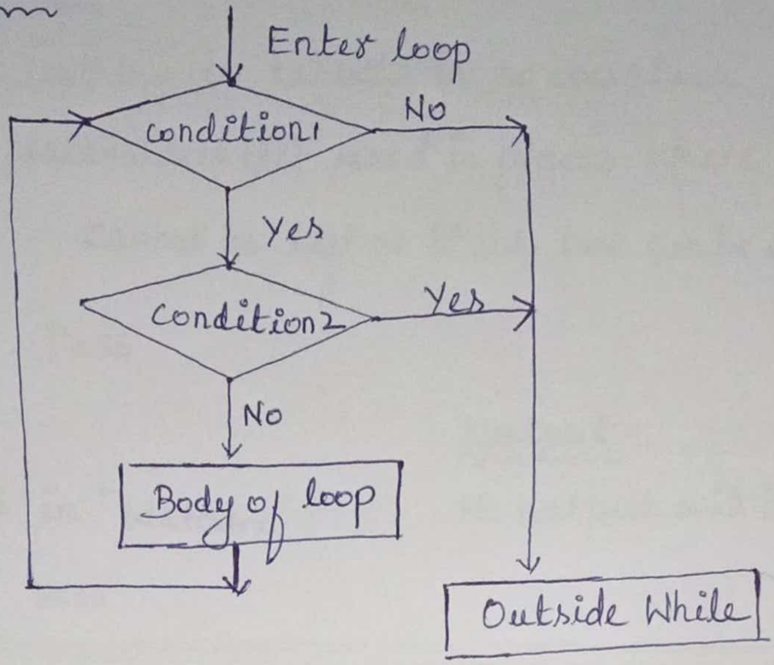
Program

```
for i in 'Welcome':
    if i=='c':
        break
    print(i)
```

O/P

```
W
e
l
```

flowchart



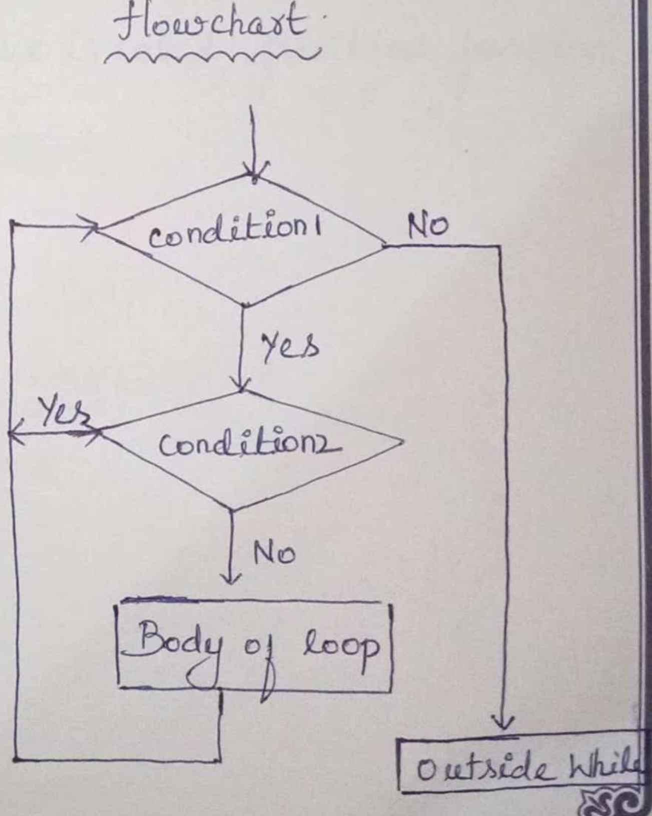
Continue.

It terminates the current iteration and transfer the control to the next iteration in loop.

Syntax

```
While condition1:

        if condition2:
                continue
```



Program

```
for i in 'Welcome':
    if i=='c':
        continue
    print(i)
```

O/P W. e l o m e

Pass statement

↳ executes Nothing (ie) results in no operation

↳ Used as placeholders (ie) used in places where the program code cannot be left as blank but can be written in future

Syntax : Pass

Program

```
for i in 'Welcome':
        pass
```

Output

```
No output will be displayed.
```

5. Fruitful function, void function, Return value, parameters & Arguments.

Fruitful function

A function that returns a value is called fruitful function.

Program.

```
def add():
    a=10
    b=20
    c=a+b
    return c

z=add()
print(z)
```

Output

```
30
```

Void function.

A function that perform action but don't return any value.

program

```
def add():
    a=10
    b=20
    c=a+b
    print(c)

add()
```

Output

```
30
```

Return values

return keywords are used to return values from function.

Eg

```
return a
return a,b
return a,b,c
return a+b
return 8
```

Parameters and Arguments : Refer unit II

6. Local & Global Scope, Function composition, Recursion

Scope.

↳ The scope of a variable refers to places that you can access a variable.

Global scope.

↳ A variable with global scope can be used anywhere in program

↳ Created by defining a variable outside the function

```
Program.                         Output

a=50
def add():
    b= 20
    c = a+b                      70
    print(c)
def sub():
    b=30
    c=a-b                        80
    print(c)
add()
Sub()
print(a)                         50
print(b)                         Error.
```

Local Scope

↳ A variable with local scope can be used only within function.

↳ Created by defining a variable inside the function.

```
    program                          Output

    def add():
        a=50
        b=20
        c=a+b                        70
        print(c)
```

```
def Sub():
        a=50
        b=30
        c=a-b
        print (c)                    20
    add()
    Sub()
    print(a)                         Error
    print(b)                         Error.
```

Function composition

↳ Function composition is a way of combining func such that result of each function is passed as argument of Next function.

Program.   Output

```
def add(a,b):                            900.
      c=a+b
      return c
def mul(c,d):
      e=c*d
      return e
 x=add(10,20)
 y=mul(x,30)
 print(y)
```

Recursion

A function calling itself till it reaches the base value - Stop point of function call.

Program: Factorial of 'n'

```
def fact(n):
    if n==0 or n==1:
        return 1
    else:
        return n*fact(n-1)

z=fact(4)
print(z)
```

Output

```
24.
```

7 String, Immutability, String functions & Methods, Modules

String.

- Collection of letter/character
- enclosed in single quote ' ' or double quote " "
- Immutable
- Indexed by integer
- String of 1 length - character.

operations.

- Creation ' ' >>> s='students'
- Indexing [ ] >>> Print(s[1])
- Slicing [:] >> print(s[2: 5])
- Repetitions * >>> print(s*2)
- concatenation + >>> print(s+'Study')

Immutability:

String is an immutable type ie its characters can be accessed but it cannot be modified.

Program:

```
s = 'Students'
S[0] = 'H'
```

O/P

```
TypeError: 's' object does not support
           item assignment.
```

✓

String functions/methods

Builtin string functions are

Eg: a = 'happy birthday'

S.No	Methods	Example	Description
1	a.capitalize()	>>> a='good ~~Day~~ day' >>>a.capitalize() 'Good day'	capitalize first letter only

2.	a.upper()	>>> a='good day'
		>>> a.upper()
		'GOOD DAY'
			convert to upper case

3.	lower()	>>>a='GOOD DAY' >>>a.lower() 'good day'	Convert to lower case
4.	title()	>>>a='good day' >>>a.title() 'Good Day'	Capitalize first letter of all Words.

5. Swapcase ( )	>>>a='GooD day' >>>a.swapcase() good DAY	Change lowercase to uppercase & vice versa
6. Split()	>>> a='good day' >>>a.split() ['good', 'day']	returns list of Words separated by space

7. Center(Width, 'filchar')	>>> a = 'good day'
	>>> a.center(12,#)
	##good day##
		pad string with specified 'filchar' till length equal to 'Width'

8. Count(Substring)	>>>a = 'good day' >>>a.count('o') 2	returns no.of occurences of substring.

9. replace(old,new)	>>> a='good day'
	>>> a.replace('day','time')
	'good time'
		replace old With new Substring.

10. join(String)	>>> b='happy'
	>>> a='_'
	>>>a.join(b)
	'h_a_p_p_y'
		returns a string concatenated with an elements of an iterable.

11. isupper ()	>>> a = 'good day'
	>>> a.isupper ()
	False
		Check for uppercase

12. islower()	>>> a = 'good day'
	>>> a. islower()
	True
		check for lowercase

13. isalpha()	>>> a = 'good day'
	>>> a. isalpha()
	False
		Check for alphabetic characters.

14. isalnum()	>>> a = 'good day'
	>>> a. isalnum()
	False
		Check for alpha-numeric characters

15. isdigit ()	>>> a = 'good day'
	>>> a. isdigit()
	False
		check for digits

16. isspace()	>>> a = ' '
	>>> a . isspace ()
	True
		check for Whitespace

17. istitle ()	>>> a = 'Good Day'
	>>> a . istitle ()
	True
		Check for titlecase

18. find(Substring)	>>> a = 'good day'
	>>> a.find ('d'))
	3
		Return index

19. Startswith(substring)	>>>a='good day' >>>a.startswith('g') True	checks Whether the string begins with substring.
20. endswith(substring)	>>> a='good day' >>>a.endswith('d') False	check Whether the string ends with substring
21. len(string)	>>>a='good day' >>> len(a) 8	Returns the length of the string
22. min(a)	>>>a='good day' >>> min(a) ' '	Returns the minimum character in string
23. max(a)	>>>a='good day' >>>max(a) 'y'	Returns the maximum character in string.

String Module

↳ Module is a file that offers additional functions, classes & variables to manipulate standard string.

↳ Import it to use in program.

Syntax :

```
import string.
```

Example

```
import string
print(string.punctuation)
print (string. digits)
print(string. printable)
print(string. capwords ('good day'))
print( string. hexdigits)
print( string. octdigits)
```

8. List as array

Array

↳ collection of elements of same type

↳ Enclosed in square brackets [ ]

↳ Mutable

↳ Indexed by integer.

Syntax to import array:

```
    import array
```

syntax to create array:

```
    array_name = array. array ('datatype', [elements])
```

Example

```
    a = array. array ('i', [1, 2, 3, 4])
```

Datatype	Description
'c'	character of size 1 byte
'b'	signed integer of size 1 byte
'B'	Unsigned integer of size 1 byte
'i'	signed integer of size 2 bytes
'I'	Unsigned integer of size 2 bytes
'l'	signed integer of size 4 bytes
'L'	Unsigned integer of size 4 bytes
'f'	floating point of size 4 bytes
'd'	floating point of size 8 bytes

program to find sum of array elements

```
import array
sum=0
a=array.array('i',[1,2,3,4])
for i in a:
        sum=sum+i

print(sum)
```

O/P

```
10
```

Convert list into array.

fromlist() → Used to convert a list into an array.

Syntax

```
        array_name.fromlist(list_name)
```

program to find sum of array elements

```
 import array
 sum=0
  l = [6,7,8,9,5]
  a = array.array('i', [])
  a.fromlist(l)
   for i in a:
          sum=sum+i

   print(sum)
```

Output

```
25
```

Methods

S.No	Syntax	Example	Description
1.	array('datatype', [elements])	>>> a=array.array ('i', [1,2,3,4,5])	Used to create an array

2.	append(element)	>>> a.append(6)
		>>> print(a)
		array('i', [1,2,3,4,5,6])
			Used to add element at end

Pros

3) insert(index, element)	>>>a.insert(2,10)
	>>>print(a)
	array('i',[1,2,10,4,5,6])
		Used to insert element at specified index

4) pop(index)	>>> a.pop(1) 2	Removes the element at index.
5) index(element)	>>> a.index(10) 1	Returns the index of valu

6) reverse()	>>> a.reverse()
	>>> print(a)
	array('i', [6,5,4,10,1])
		reverses the array

7) count(element)	>>> a.count(4) 1	Used to count the specified elements in array

Illustrative Programs.

1) Square root using Newton's method

```
num = int(input('Enter a value:'))
x = num //2
While True:
    y=(x+num /x)/2
    if x==y:
        print(y)
        break
    x=y
```

Output

```
Enter a value: 16
4.0
```

5) Linear search

```
a=[5, 4, 1, 2, 3]

x = 2

flag=0

for i in a:
        if i==x:
            flag=1
            break

if flag==1:
        print('Element found')
else:
        print('Element Not found')
```

0

Output

Element found.

6) Binary Search

```
a = [5,10,15,20,25]

x = 20

flag=0

f = 0
l = len(a)-1

while f<=l :
    mid = (f+l)//2
    if x == a[mid]:
        flag=1
        break
    else x < a[mid]:
        l =mid-1
    else:
        f =mid+1

if flag==1:
    print('Element found')

else:
    print('Element Not found')
```

Output

Element found.

2) GCD

```
a = int(input('Enter a value:'))
b = int(input('Enter b value:'))
if a<b:
    small = a
else:
    small = b
for i in range(1, small+1):
    if a%i==0 and b%i==0:
            gcd=i
    print(gcd)
```

Output

```
Enter a value: 12
Enter b value : 24
12
```

3) Exponentiation

```
a = int(input('Enter a value:'))
b = int(input('Enter bvalue:'))
z = 1
for i in range(1,b+1):
    z=z*a
print(z)
```

Output

```
Enter a value: 2
Enter bvalue: 4
16
```

4) Sum an array of elements

```
import array
a = array.array('i', [1,2,3,4])
sum=0
for i in a:
    sum = sum+i
print(sum)
```

Output

```
10
```

UNIT-IV

1) Lists : list operations, list slices, list methods, list loop, mutability , aliasing, cloning list, list parameters.

List

↳ collection of elements of different type

↳ Enclosed in square bracket

↳ Mutable.

↳ indexed by integer.

↳ values in list also called as element / item .

operations

1. Creation [ ]	>>> a=[10,20,30,40,50]
2. Indexing [ ]	>>> a[0] 10 .
3 slicing [:]	>>> print(a[1:4]) [20,30,40]
4. Repetitions *	>>> print(a*2) [10,20,30,40,50,10,20,30,40,50]
5. Concatenation +	>>> print (a+[60,70]) [10,20,30,40,50,60,70]

List slices

↳ operation that extracts a subset of elements from list and packages them as another list

Syntax

listname [start : stop [: steps]]

Here, default start value is 0, stop value is n-1.

[:] → will print entire list

[2:2] → will create an empty slice

[::-1] → Reverse the list.

Example.

```
>>> a = [10, 20, 30, 40, 50]
>>> print [ : 4].
[10, 20, 30, 40].
```

List methods

S.No	Syntax	Example	Description

1.	append(element)	>>> a = [10,20,30]
		>>>a.append(40)
		>>>print(a)
		[10,20,30,40]
			Add an element to end.

2.	insert (index, element)	>>> a = [10,20,30]
		>>>a. insert(3,40)
		>>> print(a)
		[10, 20, 30, 40]
			Insert an item at specified index

3.	extend(listname)	>>> a = [10, 20, 50]
		>>> b = [‘a’, ‘b’, ‘c’]
		>>> a. extend (b)
		>>> print (a)
		[10, 20, 30, ‘a’, ‘b’, ‘c’]
			Add second list to end of first list

4. index(element)	>>>a=[10, 20,30]. >>>a. index(20) 1	Return index of an element
5. sort()	>>>a=[7,5,9, 3] >>>a. sort() >>> print(a) [3,5,7,9]	sort in ascending order

6. reverse()	>>> a = [10,20, 30]
	>>> a. reverse()
	>>> print(a)
	[30, 20,10]
		Reverse the list

7. pop()	>>> a=[10,20,30] >>>a. pop() 30	Removes the last element

8. pop(index)	>>> a=[10,20,30].
	>>> a. pop(1)
	20
		Removes element at specified index.

9. remove(element)	>>> a=[10,20,30]
	>>> a. remove(30)
	>>> print(a)
	[10,20]
		Removes specified element from list

10. Count(element)	>>> a =[10,20,20, 30]
	>>> a.count (20)
	2
		Returns no.of occurance of an element

11. copy()	>>>a=[10,20,30] >>>b=a.copy() >>>print(b) [10,20,30]	copy the list
12. len(listname)	>>>a=[10,20,30] >>>len(a) 3	No.of elements in list
13. min(listname)	>>>a=[10,20,30] >>>min(a) 10	Returns minimum element
14. max(listname)	>>>a=[10,20,30] >>>max(a) 30	Returns maximum element

15. Clear()	>>>a=[10,20,30]
	>>>a.clear()
	>>>print(a)
	[ ]
		Removes all element

16. del(listname)	>>>a=[10,20,30]
	>>>del(a)
	>>>print(a)
	Error: name 'a' is not defined
		delete the entire list

List loops

1. For loop
2. While loop
3. Infinite loop

For loops ✓

For in list

↳ For loop in python used to iterate over sequence (list, tuple, string). Iterating over a sequence called traversal. Loop continues until we reach the last element in sequence.

↳ The body of for loop is separated from rest of code using indentation.

So	Syntax	Example	Output

1	for i in list:
	    print(i)
		for i in [1,2,3,4,5]:
		    print(i)
			1
			2
			3
			4
			5

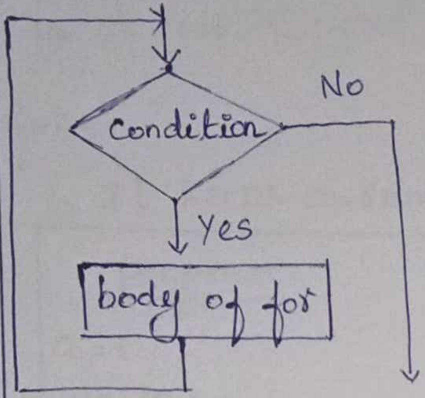
For in range

↳ To generate sequence of number, range() function is used.

↳ Used in 3 different ways.

1. range(n) # generates numbers from 0 to n-1	for i in range(5): print(i)	0 1 2 3 4
2. range(m,n) # generates numbers from m to n-1	for i in range(3,6): print(i)	3,4,5
3. range(m,n,x) # generates numbers from m to n-1 With skip x	for i in range(3,8,2): Print(i)	3,5,7

flowchart



Program : print numbers from 25 to 35

```
for i in range(25,36):
        print(i)
```

Output

```
25 26 27 28 29 30 31 32 33 34 35
```

While loop

↳ Condition is checked first

↳ If it true, body of loop is entered

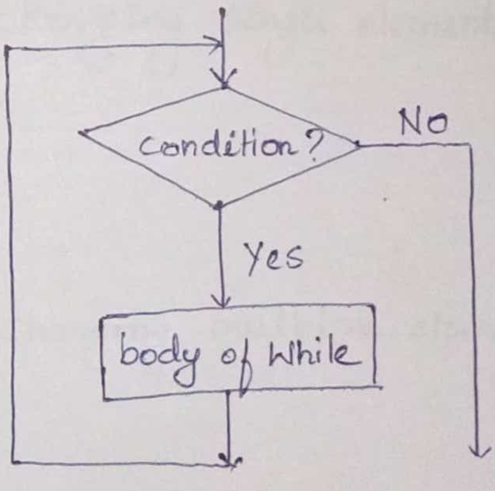
↳ After one iteration, ~~test~~ condition is checked again

↳ Process continues until the condition evaluates to False.

Syntax

```
initial value
While condition:
        body of While loop
        increment/Decrement
```

flowchart



program

print numbers from 25 to 35

```
~~for~~  i=25
     While i<=35:
          print(i)
          i=i+1
```

Output

```
25 26 27 28 29 30 31 32 33 34 35
```

Infinite loop. ✓

↳ A loop becomes infinite loop if given condition never becomes false.

↳ It keeps on running. Such loops are called infinite loop.

Program.	Output

a=1
While a==1:
    n=int(input('Enter the number:'))
    print('You entered:', n)
	Enter the number:10
	You entered:10
	Enter the number :15
	You entered :15

List Mutability

Mutable → value can be changed

Example	Description

>>> a=[1,2,3,4,5]
>>> a[0]=100
>>> print(a)
[100,2,3,4,5]
	changing single element

>>> a=[1,2,3,4,5]
>>> a[0;3]=[100,100,100]
>>> print(a)
[100,100,100,4,5]
	Changing multiple element

>>> a=[1,2,3,4,5]
>>> a[0:3]=[]
>>> print(a)
[4,5]
	Element can be removed by assigning empty list.

```
>>> a=[1,2,3,4,5]
>>> a[0:0]=[10,20,30]
>>> print(a)
[10,20,30,4,5]
```

Elements can be inserted into a list by squeezing them into an empty Slice at desired location

Aliasing.

- Creating a copy of a list is called aliasing.
- Both list will be having same memory location
- changes in one list will affect another list.

Program

```
list1 = [1,2]
list2 = list1
print (list1, list2)
list1[0]=10
print(list1, list2)
```

Output

```
[1,2] [1,2]
[10,2] [10,2]
```

Cloning

- Creating a copy of a list is called cloning
- Both list will be having different memory location
- changes in one list will not affect another list.

3 ways of cloning.

- list()
- copy()
- deepcopy()

1) list()

Syntax : Newlistname = list(oldlistname)

```
Program                                   Output
list1=[1,2]
list2=list(list1)
print(list1,list2)                        [1,2] [1,2].
list1[0]=10
print(list1, list2)                       [10,2] [1,2].
```

2) Copy()

Syntax: newlistname =copy.copy(oldlistname)

```
program                                   Output
  import copy
  list1=[1,2]
  list2=copy.copy(list1)
  print(list1, list2)                     [1,2] [1,2].
  list[0]=10
  print( list1, list2)                    [10,2] [1,2]
```

3) deepcopy()

Syntax : newlistname =copy.deepcopy(oldlistname)

```
program                                        Output
    import copy
    list1 =[1,2]
    list2 =copy.deepcopy( list1)
    print( list1, list2)                       [1,2] [1,2]
```

```
list1[0]=10
print(list1, list2)                    [10,2] [1,2]
```

List as parameters

↳ Passing a list as an argeument actually passes a reference to list, not a copy of the list.

↳ Since lists are mutable, changes made to the parameters change the argeument.

Example

```
def my_remove(a):                      Output
        a.remove(1)                    [2,3,4,5]
        print(a)                       [2,3,4,5]
a= [1,2,3,4,5]
my_remove(a)
print(a)
```

Tuple

↳ Collection of elements of different type

↳ Enclosed in paranthesis ()

↳ Immutable

↳ indexed by integer

↳ values in tuple called as element/item.

Operations

1. creation ()	>>> a= (10,20,30,40,50)
2. Indexing []	>>> print(a[3]) 40

3. slicing [:]	>>> print(a[1:]) [20,30, 40,50]
4. repetitions *	>>> print(a*2) (10,20,30,40, 50, 10, 20,30, 40,50)
5. concatenation +	>>> print(a+(‘a’,‘b’)) (10,20,30,40,50,‘a’,‘b’)

Tuple methods.

a=(10,20,30,40,50)

1. index(element)	>>> a.index(50) 4	Returns index of element
2. count(element)	>>> a.count(50) 1	Returns no.of occurrence of element
3. len(tuplename)	>>> len(a) 5	Returns length of tuple
4. min(tuplename)	>>> min(a) 10	Return minimum element
5. max(tuplename)	>>> max(a) 50	Return maximum element
6. del(tuple)	>>> del (a)	delete entire tuple

Tuple Assignment

↳ allows, variables on left of an assignment operator & values on right of an assignment operator.

↳ Uses.

often useful to swap values of 2 variables

Program : Swapping

```
a = 20
b = 50 .
(b,a) = (a,b)
print(a,b)
```

Output

```
50, 20 .
```

Multiple Assignments

↳ assign multiple values to multiple variables.

program

```
(a,b,c) = (10,20,30)
print(a,b,c)
```

Output

```
(10,20,30)
```

Tuple as ~~multiple assignment~~ return value

↳ A function can return one value.

↳ More than one value from a function, can be returned by using tuple.

program

```
def div(a,b):
    q = a//b
    r = a%b
    return (q,r)

a=10
b =2
q,r = div (a,b)

print(q,r)
```

Output

```
5 0
```

Tuple as arguments

↳ parameter name that begins with * gathers argument into a tuple.

program

```
def display(*args):
      print(args)
display(2,3,'a')
```

Output

```
(2,3,'a')
```

3. Dictionaries: operations and methods.

Definition

↳ collection of values of different type.

↳ Enclosed in curly braces { }.

↳ Mutable

↳ indexed by key.

↳ Store in key-value pair.

	Operation	Code key value	Comment
1.	Creation { }	>>> a = {1:'a', 2:'b'}	Creating directory.
2.	Access an element	>>> print (a[2]) 'b'	Return value of key

3.	Update an element	>>> a[1] = 'apple'
		>>> a
		{1:'apple', 2:'b'}
			Update value of key

4.	Add an element	>>> a[3] = 'c'
		>>> a
		{1:'apple', 2:'b', 3:'c'}
			Add new value

5.	Delete an element	>>> del (~~dict~~ a[1])	Delete desired key-value pair from dictionary.
6.	Delete an entire directory	>>> del (a)	Delete all items in dictionary

Methods

Methods	Description	~~Syntax~~ Example.

dict()	Creates a new dictionary	>>> d=dict()
		>>> print(d)
		{}

dict(dict_name)	Copy a dictionary	>>> t={1:'a', 2:'b'}
		>>> d=dict(t)
		>>> print(d)
		{1:'a', 2:'b'}

len(dict_name)	Return no.of items	>>> d={1:'a', 2:'b'}
		>>> len(d)
		2

items()	return items of dictionary	>>> d={1:'a', 2:'b'}.
		>>> print(d.items())
		dict_items([(1,'a'), (2, 'b')])

keys()	returns Keys of dictionary	>>> d={1:'a', 2:'b'}
		>>> print(d.keys())
		dict_keys ([1,2])

values()	returns value of dictionary	>>> d={1:'a', 2:'b'}
		>>> print(d.values()
		dict_values (['a', 'b'])

pop(key)	Remove element on key	>>> d={1:'a', 2:'b'}
		>>> d.pop(1)
		'a'.

popitem()	Remove last element	>>> d={1:'a', 2:'b'}.
		>>> d.popitem()
		(2:'b')

clear()	Remove all element	>>> d={1:'a', 2:'b'}
		>>> d.clear()
		>>> d
		{}

get(key)	Return element from key	>>> d={1:'a', 2:'b'}
		>>> d.get(2)
		'b'.

Sorted(dict_name)	sort the elements	>>> d={1:'a', 4:'d', 2:'b'}
		>>> sorted(d)
		[1, 2, 4]

key in d	Check whether key in dictionary	>>>d={1:'a', 2:'b'} >>> print(2 in d) True.

4. Advanced list Processing - list comprehension

list comprehension.

↳ Comprehension are constructs that allows sequences to be built from other sequences.

↳ can be used for filtering.

4 parts

↳ an input sequence.

↳ Variables representing members of input sequences.

↳ An optional predicate expression.

↳ An output expression producing elements of output list

Syntax

```
[expression for item in list if conditional]
```

This is equivalent to:

```
for item in list :
    if condition:
        expression.
```

Example 1:

```
>>> [i**2 for i in range (5) if i==4]
[16]
```

Input sequence : range(5)

Variables representing members of input sequences : i

Optional predicate expression : i==4

Output expression : i**2.

Example 2:

```
>>> [i**2 for i in range(5) if i%2==0]
[0,4,16]
```

Input sequence : range (5)

Variables representing members of input sequences : i

Optional predicate expression : i%2==0

Output expression : i**2.

Illustrative programs:

Simple Sorting - Selection sort

```
a = [23, 41, 17, 3, 20]
for i in range(0, len(a)):
    pos=i
    for j in range( i+1, len(a)):
        if a[pos]> a[j]:
            pos=j
```

O/p

```
[3, 17, 20, 23, 41]
```

```
        temp= a[i]
        a[i] = a[pos]
        a[pos] = temp
print(a)
```

Insertion Sort:

```
a = [5, 7, 1, 4, 3]
for i in range (1, len(a)):
        temp= a[i]
        j = i-1
        while j>=0 and a[j]>temp:
                a[j+1] = a[j]
                j = j-1
        a[j+1]=temp
print (a)
```

O/P

```
    [1, 3, 4, 5, 7].
```

Merge Sort

```
def  mergesort(alist):
    if len(alist)>1:
        mid= len(alist) //2
        lefthalf =alist [: mid]
        righthalf= alist [mid:]
        mergesort ( lefthalf)
        mergesort (righthalf)
        i=0
        j=0
        k=0
        While  i<len(lefthalf) and j<len(righthalf):
            if  lefthalf[i] <righthalf[j]:
                alist[k] = lefthalf[i]
                i = i+1
            else;
                alist [k] = righthalf [j]
                j = j+1
            k=k+1
        While  i< len(lefthalf):
            aalist [k] = lefthalf[i]
            i = i+1
            k= k+1

        While  j< len( righthalf):
            alist [j] = righthalf [j]
            j=j+1
            k= K+1
        print (alist)

alist = [5,7,1,12,2]
mergesort (alist)
print(alist)
```

UNIT-V

FILES

7

1.1 File

A file is a collection of data stored in a particular area on disk. To keep the data permanent, files are used.

Two types of file

- Text file
- Binary file

Binary file

- computer can understand
- returns bytes when reading from file
- Eg: image, or exe files

Text file

- Stored on harddisk or CD-ROM
- containing characters, structured as individual lines of text.
- contain non-printing newline character \n
- Viewed and created using text editor.
- In python, default file type is text file

1.2 file modes

Modes	Description
r	Opens a file for reading only.
W	opens a file for writing only
a	opens a file for appending only.
r+	opens a file for both reading and writing
W+	opens a file for both writing and reading
a+	opens a file for both appending and reading
rb	opens a file for reading in binary format
Wb	opens a file for writing in binary format
ab	opens a file for appending in binary format
rb+	opens a file for both reading & Writing in binary format
Wb+	opens a file for both writing & reading in binary format
ab+	opens a file for both appending & reading in binary format

1.3 file operations

A file operation takes place in the following order

1. open a file
2. Read or write (perform operation)
3. Close the file.

1. Opening a file ✓

- Open a file before use
- open() - used to open a file
- returns file object

Syntax

```
fileobject =open(filename, mode)
```

Eg

```
f=open('output.txt','W')
```

2. Reading and Writing ✓

Read operation

- read() - used to read the contents from file
- open file in 'r' mode

Syntax

```
fileobject = open(filename,'r')
fileobject.read()
```

Eg

```
>>> f=open('output.txt', 'r')
f.read()
```

Write operation

- Write() - used to write data into the file
- open file in 'W' mode

Syntax

```
fileobject = open(filename, 'w')
fileobject.write(string)
```

Eg

```
>>> f=open('output.txt', 'W')
f.write('python')
```

3. Closing a file ✓

↳ close the file after use

↳ close() - used to close the file

Syntax

```
Fileobject.close()
```

Example

```
>>> f.close()
```

1.4 File object Attributes

fileobject.closed - returns true if the file is closed, Otherwise false

fileobject.mode - returns the file access mode

fileobject.name - returns the name of the file

program

```
f = open('file.txt', 'w')
print(f.name)
print(f.mode)
print(f.closed)
```

Output

```
file.txt
w
False
```

1.5 File Methods

1. close() - close a file after it has been open
2. read() - Read the content of the file
3. readable() - check if the file is readable
4. readline() - read the first line of the file
5. readlines() - return all lines in the file as a list Where each line is an item.

6. Write () - Write the specified line to the file

7. Writable() - Check if the file is writable

8. Writelines() - Write a list of lines to the file

9. Seek () - change the current file position

10. Seekable() - check if the file is seekable

11. tell() - Return the current file position.

12. truncate(size) - Resize the file to the given no. of bytes

13. fileno() - Written the file descriptor of the stream

14. flush() - Clear the internal buffer

15. isatty() - check if the file is connected to a terminal device

Program

```
f=open('file.txt','w')
f.write("Python is a programming
                 language")
f.close()
f=open('file.txt','r')
str = f.read()
print(str)
f.close()
```

Output

```
Python is a programming
            language
```

1.6 Format operator

The Write() accept only string as arguments. To give other values in Write(), the values must be converted to string.

Method-I : str()

str() - used to convert other values to string.

program

```
f = open('file.txt', 'w')
f.write(str(100))
f.close()
f = open('file.txt', 'r')
str = f.read()
print(str)
f.close()
```

output

```
100
```

Method-II : Format operator %.

The first operand contains one or more format sequences (%d, %g, %s) Which specify how the second operand is formatted. The result is string.

Format sequence	Description	Usage
%d	Second operand → decimal	'%d' % 100
%g	second operand → floating point	'%g' % 10.1
%s	second operand → string	'%s' % 'Hai'

Program

```
f = open('file.txt', 'W')
f.write('%d' % 150)
f.close
f = open('file.txt', 'r')
str = f.read()
print(str)
f.close()
```

Output

```
150.
```

2. Command Line Arguments

↳ Command line arguments are arguments passed into the program from the command line prompt.

↳ Supported by sys module

↳ Import sys module in program

↳ sys.argv → contains command line arguments.

↳ len(sys.argv) → finds the no. of command line arguments

Program:

Find the occurrence of each word (word count) in a text file using command line arguments

Ex10.py

```
import sys
if len(sys.argv) != 2:
        print ('Error')
        Sys.exit(1)
filename = sys.argv[1]
f=open(filename,'r')
wordcount = {}

for word in file.read().split():
    if word in wordcount:
        wordcount[word] + = 1
    else:
        wordcount [word] = 1

print (wordcount)
```

file.txt

```
VV College
VV Staffs
VV Students
```

Run as (in windows OS)

```
>python Ex10.py file.txt
VV 3
College 1
Staffs 1
Students 1
```

3. Error and Exceptions

Error :

↳ Also called as bugs

↳ mistake made by programmer

↳ Types

logical Error

Syntax Error

logical Error

↳ Error occurs due to logical mistake of program

Examples.

1. Using integer division instead of floating point division

2. Wrong intendation.

Program : Find factorial of a number 5

```
i=1
fact=0
While i<=5:
    fact=fact*i
    i=i+1
print(fact)
```

Output

```
0 ×
```

Syntax Error : Error occurs due to invalid syntax.

Example

1. Misspelling a keyword
2. Missing colon, comma or brackets

Program

```
a = int(input('Enter a number')
print(a)
```

Output

```
Invalid syntax.
```

2.1 Exceptions

↳ An exception is an error that occurs during execution of a program.

↳ also called as runtime errors.

Eg

1. Division by zero (ZeroDivisionError)
2. Using an undefined identifier (NameError)

Program

```
a = 10
print(b)
```

Output

```
NameError: b is not defined.
```

Standard Exceptions Available in python

1. ArithmeticError - Raised When an arithmetic calculation error occurs

2. ZeroDivisionError - Raised When dividing by zero

3. TypeError - Raised When an invalid operation applied to datatype.

4. keyError - Raised When specified key not found in dictionary

5. IndexError - Raised When specified index not found in sequence

6. NameError - Raised When using an undefined identifier.

7. ValueError - Raised When an inappropriate value is given

8. IOError - Raised When an input/output operation fails.

4. Handling Exceptions

↳ contains try and except block.

↳ Try block - contains code that may create exception

↳ except block - contains code that handle exception.

Types.

1. try.... except
2. try.... except inbuilt exception
3. try.... except.... else
4. try.... except.... else.... finally
5. try.... except.... except
6. try.... raise.... except
7. User defined Exception.

1) try.... except

Syntax

```
try:
    Statements
except:
    If there is Exception , this block gets executed
```

Program

```
try:
  age=int(input('Enter age'))
  print(age)
except:
   print('Invalid age')
```

Output

```
Enter age: 10
10

Enter age: a
Invalid age.
```

2) try.... except inbuilt exception

Syntax

```
try:
    statements
except  inbuilt exception:
      If there is exception, this block gets executed.
```

Program

```
try: age=int(input('Enter age'))
     print(age)
except ValueError:
   print('Invalid age')
```

Output

```
Enter age: 10
10

Enter age: a
Invalid age
```

3) try... except... else

Syntax

```
try:
    statements
except Exception:
    If there is Exception, then execute this block
else:
    No Exception, then execute this block
```

Program.

```
try:
    age=int(input('Enter age'))
except ValueError:
    print('Please type a number')
else:
    print(age)
```

Output

```
Enter age : six
please type a number

Enter age:6
6
```

4) try... except... else... finally.

Syntax.

```
try:
    statements
except Exception:
    If Exception, then execute this block
else:
    No Exception, then execute this block
Finally:
    statements to be executed always
```

Program

```
try:
    age=int(input('Enterage:"))
except ValueError:
    print('Invalid input')
else:
    print(age)
finally:
    print('Bye')
```

Output

```
Enter age:10
10
Bye

Enter age:10
Invalid input
Bye
```

5) try...except...except.

Method1

```
try:
    statements
except   Exception1, Exception2, ...Exception N
        If   Exception, execute this block
```

Program

```
try:
    a=int(input('Enter a:'))
    b=int(input('Enter b:'))
    c=a/b
    print(c)
except  ValueError, ZeroDivisionError:
    print('Exception occur')
```

Output

```
Enter age: 10
Enter  b : h
Exception occur

Enter a : 10
Enter b : 0

Exception occur
```

Method 2:

```
try:
    statements
except inbuilt exception:
        Exception handling statement
except inbuilt exception:
        Exception handling statement
```

Program

```
try:
    a=int(input('Enter a:'))
    b=int(input('Enter b:"))
    c=a/b
    print(c)
except ValueError:
        print('Invalid input')
except ZeroDivisionError:
        print('can't divide by Zero')
```

Output

```
Enter a: 10
Enter b: h
Invalid input

Enter a: 10
Enter b: 0
Can't divide by Zero
```

try...raise...except

Syntax

```
try:
    ......
    raise errorname
```

except inbuilt exception:

Exception handling statement.

Program.

```
try:
    age = int(input('Enter age'))
    if age <0:
        raise ValueError

    print(age)
except ValueError:
    print('Invalid input')
```

Output

```
Enter age: -6
Invalid input
```

User-defined Exceptions

Programmers may name their own exceptions by creating a new exception class. Exception need to be derived from the Exception class, either directly or indirectly.

Example

program

```
Class Error(Exception):
    'Base class for other exception'

pass
```

```
Class ValueTooSmallError(Error):
   'Raised When input value is too small'
pass
class ValueTooLargeError(Error):
    'Raised When input value is toolarge'
 pass

 number =10
 while True:
     try:
         x = int(input('Enter x value'))
         if x < number:
              raise ValueTooSmallError
         elif x > number:
              raise ValueTooLargeError
         break
     except ValueTooSmallError:
          print('Too Small')
          print()
      except ValueTooLargeError:
          print('Too Large')
          print().
print('Guessed correctly')
```

Output

```
Enter x value:2

Too Small

Enter x value:15

Too Large

Enter x value:10

Guessed correctly
```

5. Modules.

↳ Module is a file containing python definitions, functions, statements and instructions.

↳ Standard library of python is extended as Modules

↳ import modules to use it in programs.

↳ help(module_name) -> to get information about functions and variables in modules.

↳ dir(math) -> To list functions + variables in modules.

OS module

↳ provides function for interacting with OS.

↳ Import OS. module to use it.

Syntax

```
import os
```

Method	Example	Description
name	os.name	This function gives the name of operating system
environ	os.environ	Get users environment
getcwd()	os.getcwd()	returns current working directory
mkdir(folder)	os.mkdir('python')	create directory
rename(old_name, new_name)	os.rename('python', 'pspp')	Rename directory

remove (folder)	os.remove('PSPP')	Remove directory

Sys module. ✓

provide access to variables, methods used by interpreter.

Syntax.

```
import sys
```

Method	Example	Description
sys.argv	sys.argv	provides list of command line arguments.
	sys.argv[0]	give text file name
	sys.argv[1]	give python file name
Sys.path	sys.path	provides search path for modules
Sys.path.append()	sys.path.append()	provides access to specific path to program
sys.platform	sys.platform	provides info about OS platform
sys.exit	sys.exit	Exit from python

Steps to create own modules.

program: calculator.py

```
def add(a,b):
    print(a+b)
def ~~add~~ sub(a,b):
    print(a-b)
def mul(a,b):
    print(a*b)
def div(a,b):
    print(a/b)
```

Output

```
>>>import calculator.
>>>calculator.add(10,5)
15
>>>calculator.sub(10,5)
5
>>>calculator.mul(10,5)
50
>>>calculator.div(10,5)
2
```

Packages.

↳ A package is a collection of python modules

↳ Module is a single file containing function definitions.

↳ A package is a directory (ie folder) of python modules containing an additional __init__.py file to differentiate package from directory.

↳ __init__.py is a directory indicates to interpreter that directory should be treated like package.

Steps to create package.

Step1 : create package directory.

↳ Create a directory (ie calculator), treated as package.

step2 : Write modules for directory.

↳ Add the modules add, sub, mul, div in calculator directory.

```
                                                  div.py
                sub.py            mul.py
: add.py
 def add(a,b):   def sub(a,b):   def mul(a,b):   def div(a,b)
     print(a+b)      print(a-b)      print(a*b)      print(a/b)
```

Step3 : Add __init__.py file in calculator directory

```
    __init__.py
from .add import add
from .sub import sub
from .mul import mul
from .div import div.
```

Step 4 :

↳ Import calculator package in program

↳ Add path of package (ie "c:\python34") in program by using sys.path.append()

Program : output.py.

```
import calculator
import sys
sys.path.append("C:\python34")
print(calculator.add(5,7))
```

Output

```
12.
```

Illustrative program

1. Word count by using command line argument

```
import sys
  if len(sys.argv) != 2:
      print('Error')
      Sys.exit(1)
filename = sys.argv[1]
file = open(filename, 'r')
Wordcount = {}
for word in file.read().split():
        if word not in wordcount:
              wordcount[word] = 1
        else:
              wordcount[word] + = 1
print(wordcount)
```

file.txt

```
Hi students
Hi students of VV
```

Cmd.exe

```
C:\python37-32> python Ex10.py
          file.txt
Hi  2
students  2
of   1
VV   1
```

2) copy and display file

```
f1 = open('1.txt','r')
f2 = open('2.txt','wt')
for i in f1:
        f2.write(i)
f2.seek(0)
print(f2.read())
f1.close()
f2.close()
```

Output

```
1.txt

hello
Welcome to python program

2.text

hello
Welcome to python program
```