



PIE Tech

POLLACHI INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Approved by **AICTE** and Affiliated to **Anna University**)

sky is the limit

Department of Computer Science And Engineering

Regulation 2021

I Year – I Semester

GE3151 Problem Solving and Python Programming

Unit I

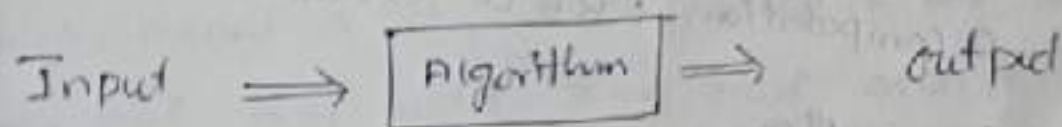
Algorithmic Problem Solving

1.1 Algorithms

↳ The term algorithm was derived from the name of Mohammed al-Khwarizmi, a Persian mathematician in the ninth century.

Al-Khwarizmi → Algorismus (in Latin) → Algorithm

- An algorithm is a 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.



1.1.1 The characteristics of a good Algorithm

- Precision
- Uniqueness
- Finiteness
- Effectiveness
- Input
- Output
- Generality

1.2. Building Blocks of Algorithm

- An algorithm is an effective method that can be expressed within a finite amount of space and time and in a well-defined formal language for calculating a function.

- Starting from an initial state and initial input, the instructions describe a computation that, when executed for it to turn green. She will check the lights and wait, check and wait. This sequence will be repeated until the lights turn green.

1.2. An instruction is a single operation that describes a computation. When it is executed it converts one state to another.

The state of an algorithm is defined as its condition regarding stored data.

- Control flow is the order in which individual statements, instructions or function calls of an algorithm are executed or evaluated.

- For complex problems our goal is to divide the task into smaller and simpler functions during algorithm design. So a set of related sequence of steps, part of larger algorithm is known as functions.

1.2.1 Control Flow

- Algorithm has three control flows

- Sequence

- Selection

- Iteration

Sequence

- A sequence is a series of steps that occur one after the other, in the same order every time.

Selection

- A selection is a decision that has to be made.

Iteration

- Iteration is sometimes called repetition, which means a set of steps which are repeated over and over until some event occurs.

1.2 Notation of Algorithm

There are four different Notation of representing algorithms

- step-form

- pseudocode

- Flowchart

- programming Language

1.3.1 Algorithm as stop-form

It is a step-by-step procedure for solving a task or problem. The steps must be ordered, unambiguous and finite in number. It is English like representation of logic which is used to solve the problem.

Step 1: Start

Step 2: Read the three number A, B, C

Step 3: Compare A and B. If A is greater perform step 4 else perform step 5.

Step 4: Compare A and C. If A is greater, output "A is greater" else output "C is greater". Then go to step 6.

Step 5: Compare B and C. If B is greater, output "B is greatest" else output "C is greatest".

Step 6: Stop.

1.3.2 Pseudocode

- Pseudocode ("sort of code") is another way of describing algorithms. It is called "pseudo" code because of its strong resemblance to "real" program code.

- Pseudocode is essentially English with some defined rules of structure and some keywords that make it appear a bit like program code.

1.3.3 Flowcharts

Flowcharts are a diagrammatic method of representing algorithms. They use an intuitive scheme of showing operations in boxes connected by lines and arrows that graphically show the flow of control in an algorithm.

Symbol	Name	Description
	Flowline	Flow line connectors show the direction that the process flows.
	Terminator	Terminators show the start and stop points in a process.
	Data (I/O)	The Data flowchart shape indicates inputs to and outputs from a process. As such, the shape is more often referred to as an I/O shape than a Data shape.
	process	show a process or action step. This is the most common symbol in flowcharts.
	Decision	Indicates a question, a branch in the process flow.

1.3.4 Control Structures as pseudocode and Flowcharts

Each control structures can be built from the basic elements as shown below.

Sequence: A sequence is a series of steps that take place one after another. Each step is represented here by a new line.

Pseudocode

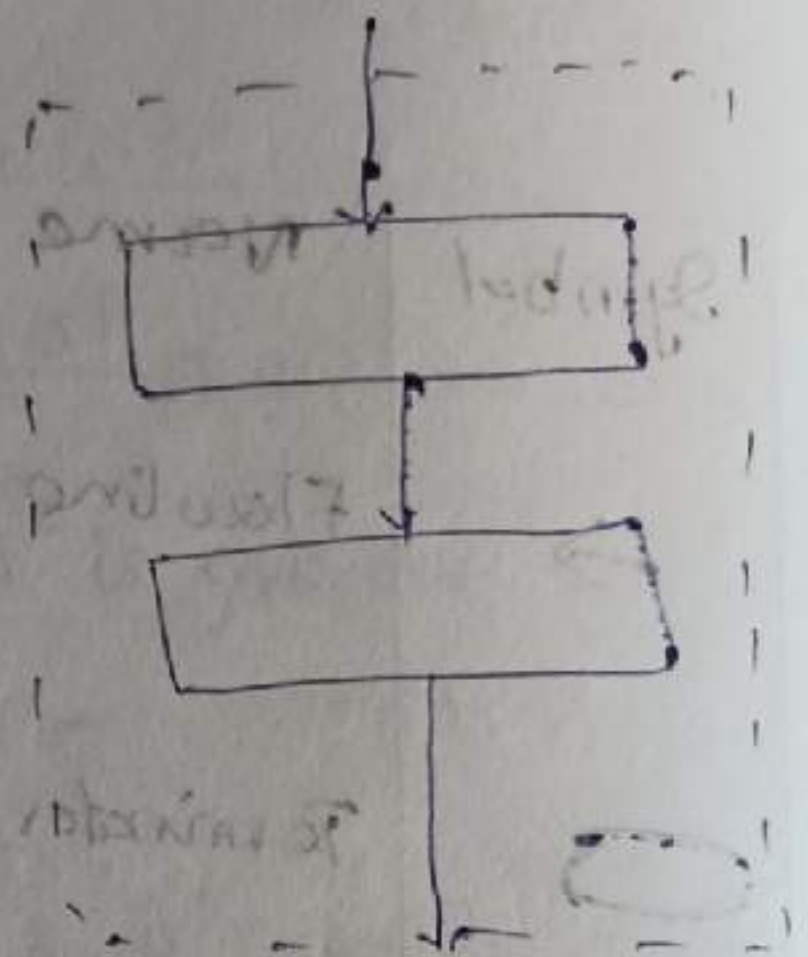
Flowchart

Begin

Statement

Statement

END



Selection: A selection is a decision. Some decisions may be answered as yes or no. These are called binary selections. Other decisions have more than two answers. These are called multiway selections.

Pseudocode

Flowchart

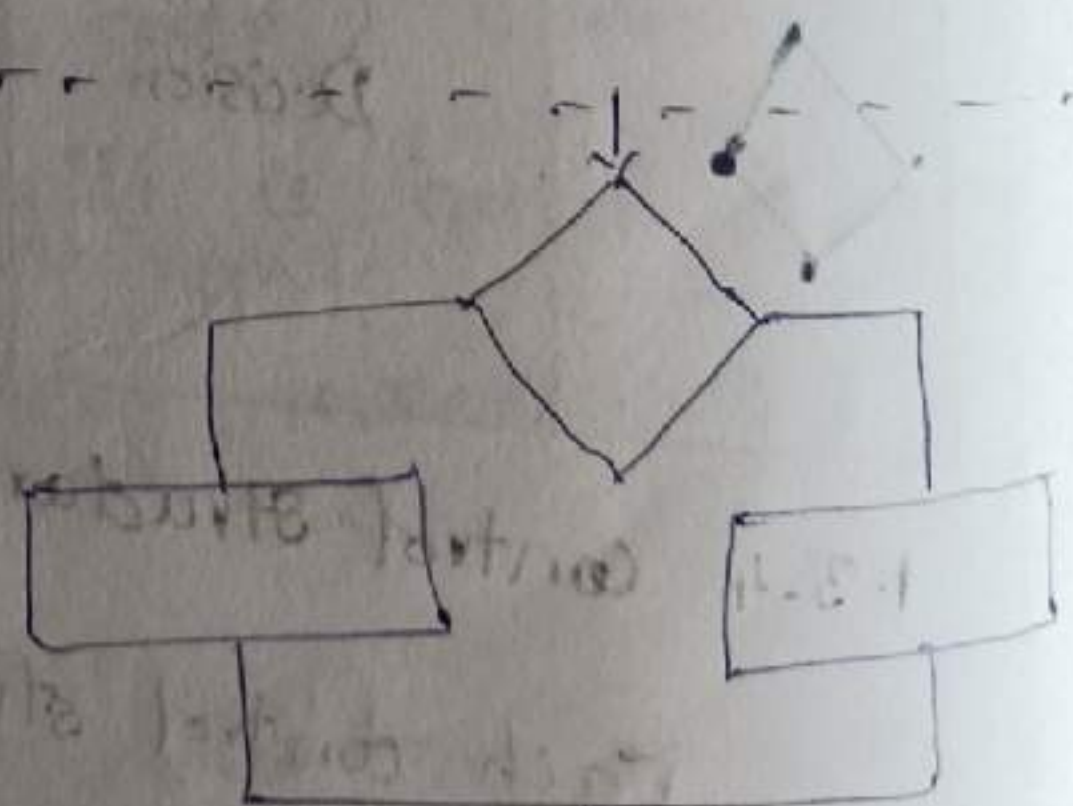
IF (question) THEN

Statement

ELSE

Statement

ENDIF



Repetition:

A sequence of steps which are repeated a number of times, is called repetition. For a repeating process to end, a decision must be made. The decision is usually called a test.

pre-test repetitions will perform the test before any part of the loop occurs.

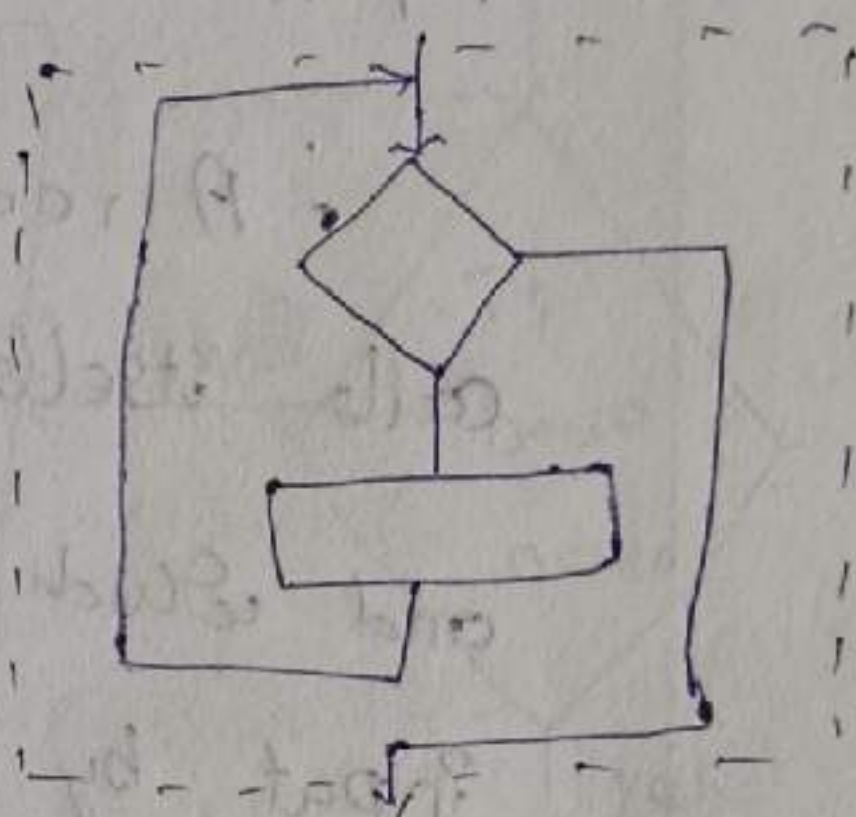
Pseudocode

WHILE (question)

statement

ENDWHILE

Flowchart



Post-test repetitions will perform the test after the main part of the loop (the body) has been performed at least once.

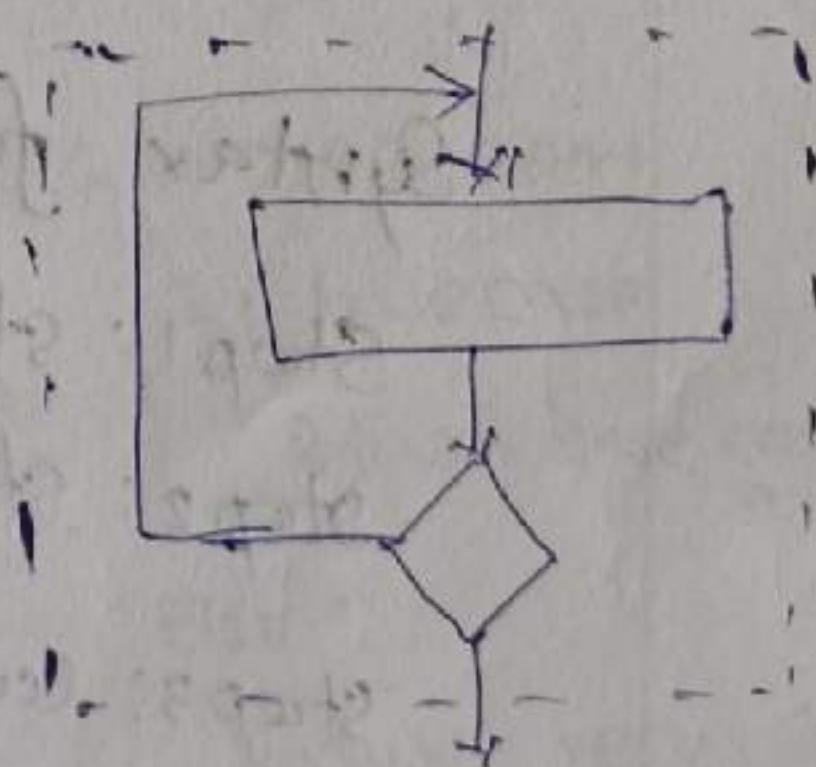
Pseudocode

REPEAT

statement

UNTIL (question)

Flowchart



1.3.5 Programming Language

Computers won't understand your algorithm as they use a different language. It will need to be translated into code, which the computer will then follow to complete a task. This code is written in a programming language.

1.4 Recursion

A recursive algorithm is an algorithm which calls itself with "smaller (or simple)" input values, and which obtains the result for the current input by applying simple operations to the returned value for the smaller input.

Base case is the case for which the solution can be stated non-recursively (i.e. the answer is known).

Recursive case is the case for which the solution is expressed in terms of a smaller version of itself.

Syntax for recursive algorithm

Step 1: start

Step 2: statement

Step 3: call subprogram

Step 4: statement

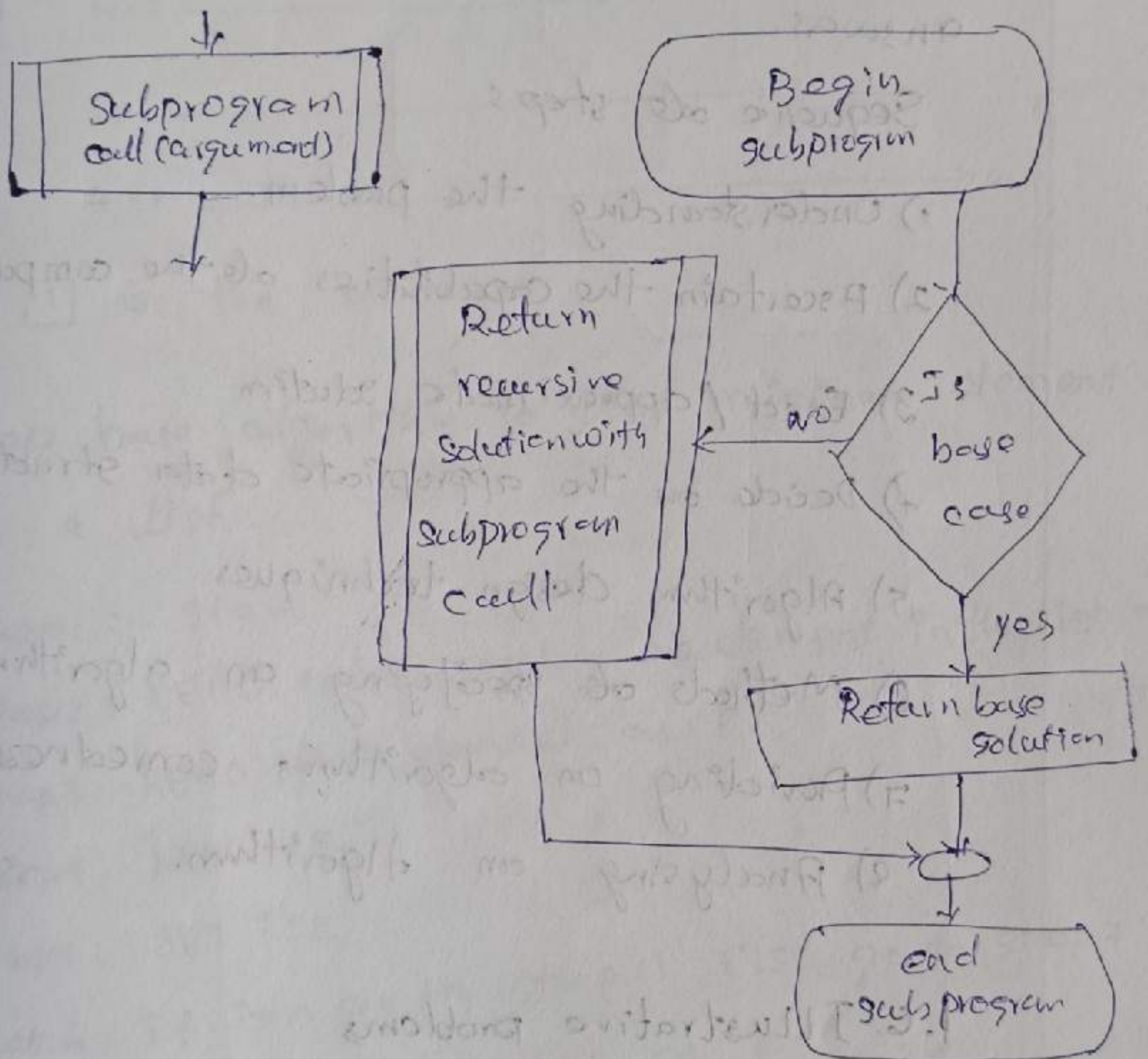
Step 5: stop

Step 1: BEGIN subprogram

Step 2: If base case then return base solution

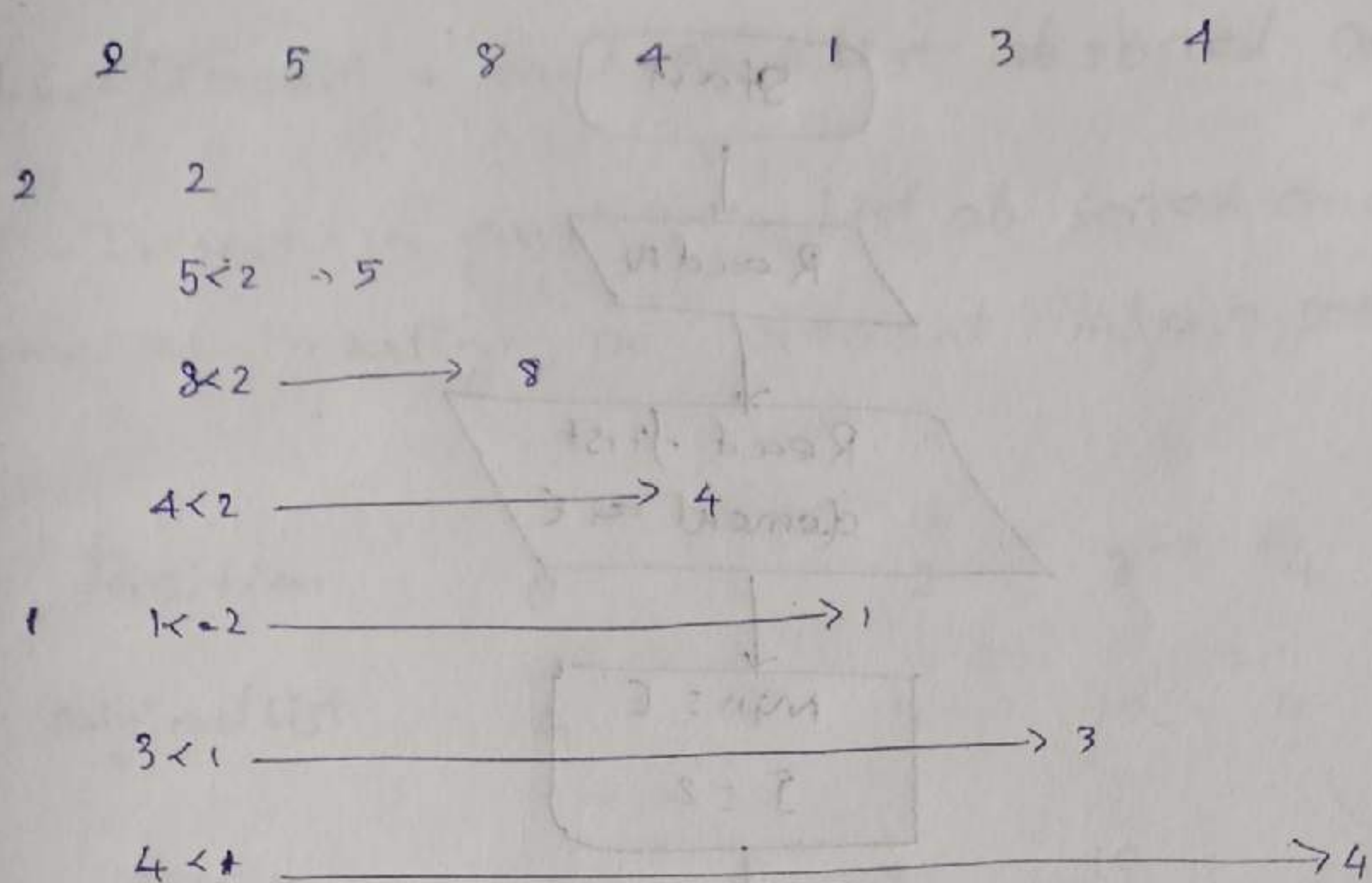
Step 3: Else return recursive solution, include
call subprogram

1.4.3. Recursion in Flow chart



1.4.2. Recursion in Pseudocode

```
SubprogramName(argument)  BEGIN SUBPROGRAM
    Is base case
    Return base case solution
Else
    Return recursive solution with subprogram name
ENDSUBPROGRAM
```



1 is the minimum

Step base algorithm to find minimum element in a list.

Step 1: Start

Step 2: READ total number of element in the List as n

Step 3: READ first element as E .

Step 4: $MIN = E$

Step 5: SET $i = 2$

Step 6: If $i > n$ go to step 11. ELSE go to step 7

Step 7: READ i th element as E

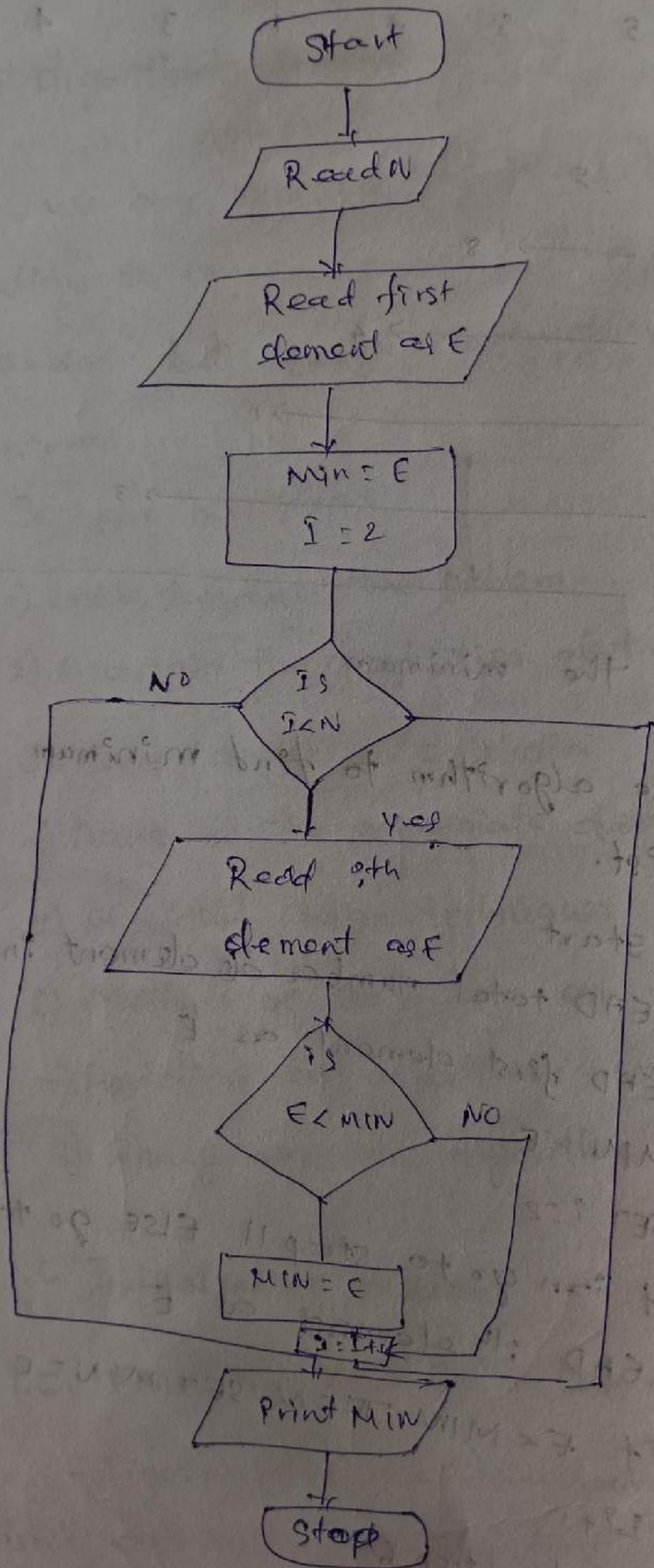
Step 8: If $E < MIN$ THEN, SET $MIN = E$

Step 9: $i = i + 1$

Step 10: go to step 6

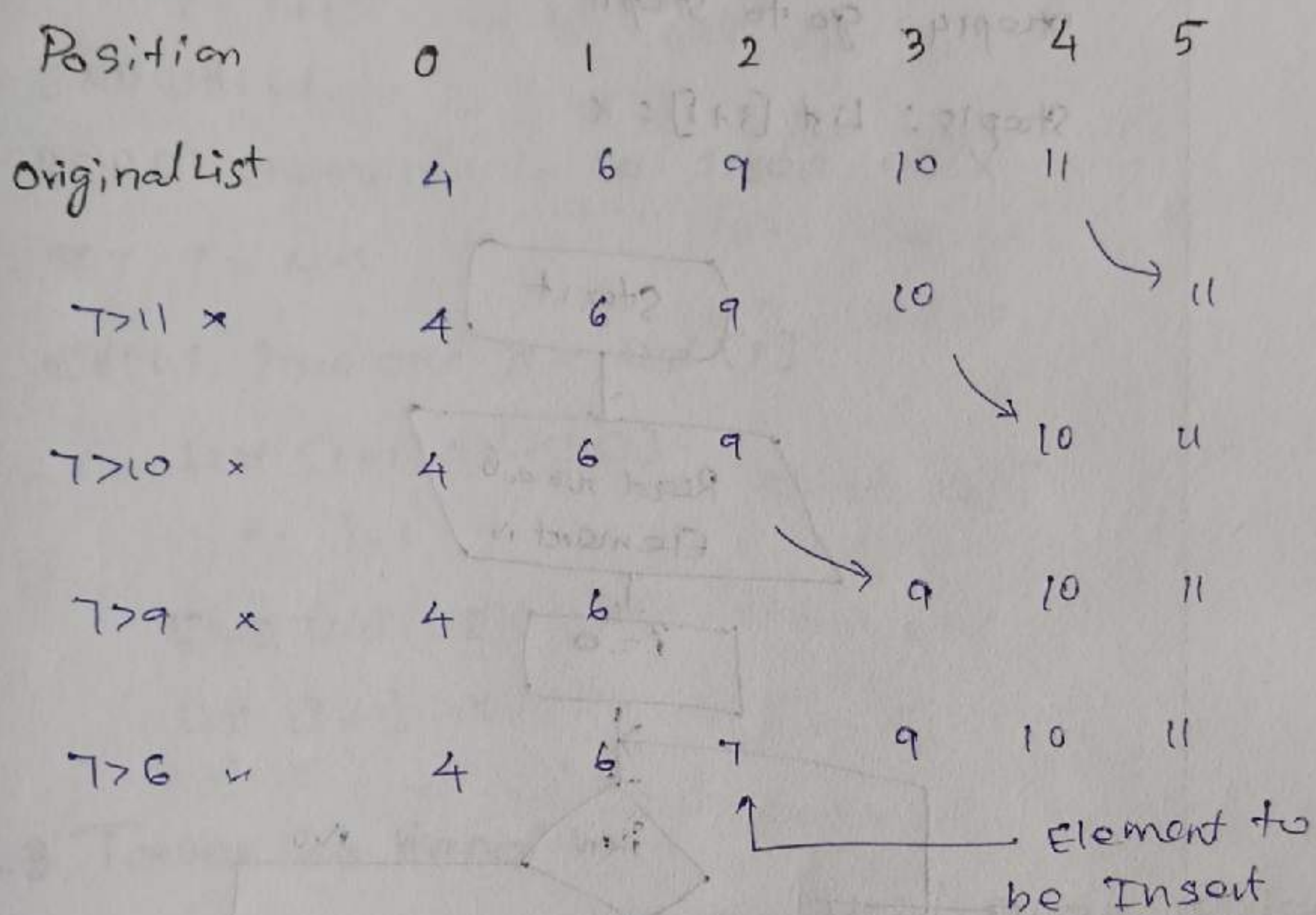
Step 11: print MIN

Step 12: Stop



1.6.2 Insert a card in a List of sorted Cards

Insert a card in a List of sorted cards - is same as inserting an element into a sorted array.



Algorithm

Step 1: start

Step 2: Declare variables N , $List[]$, i , and x

Step 3: READ Number of elements in sorted List as N

Step 4: SET $i = 0$

Step 5: IF $i < N$ THEN go to step 6 ELSE go to step 9

Step 6: READ sorted List element as $List[i]$

Step 7: $i = i + 1$

Step 8: go to step 5

Step 9: READ element to be insert as x .

Step 10: SET $i = N - 1$ THEN go to step 12

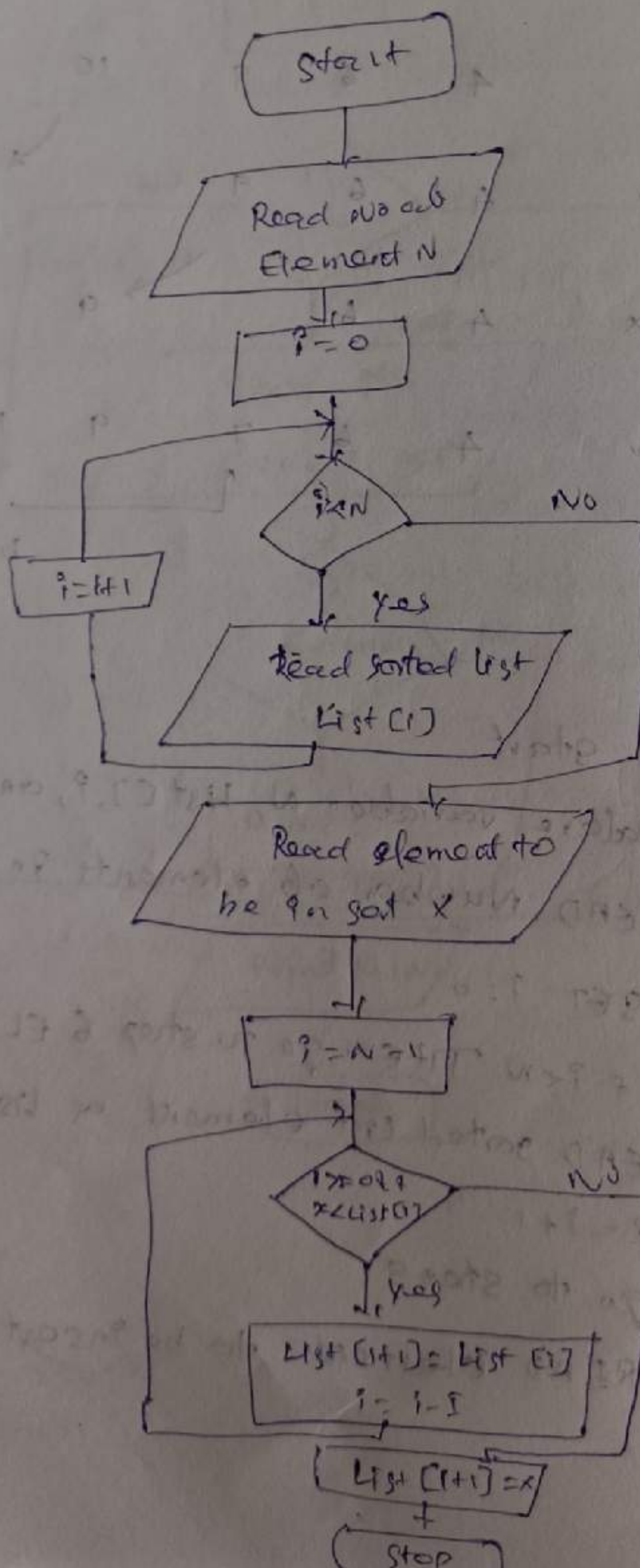
Step 11: If $i > 0$ AND $x < \text{List}[i]$ THEN go to step 15 ELSE go to step 12

Step 12: $\text{List}[i+1] = \text{List}[i]$

Step 13: $i = i - 1$

Step 14: go to step 11

Step 15: $\text{List}[i+1] = x$



Pseudocode

READ Number of element in sorted list as N

SET $i = 0$

WHILE $i < N$

READ sorted list element as $List[i]$

$i = i + 1$

ENDWHILE

READ Element to be insert as X

SET $i = N - 1$

WHILE $i >= 0$ and $X < List[i]$

$List[i+1] = List[i]$

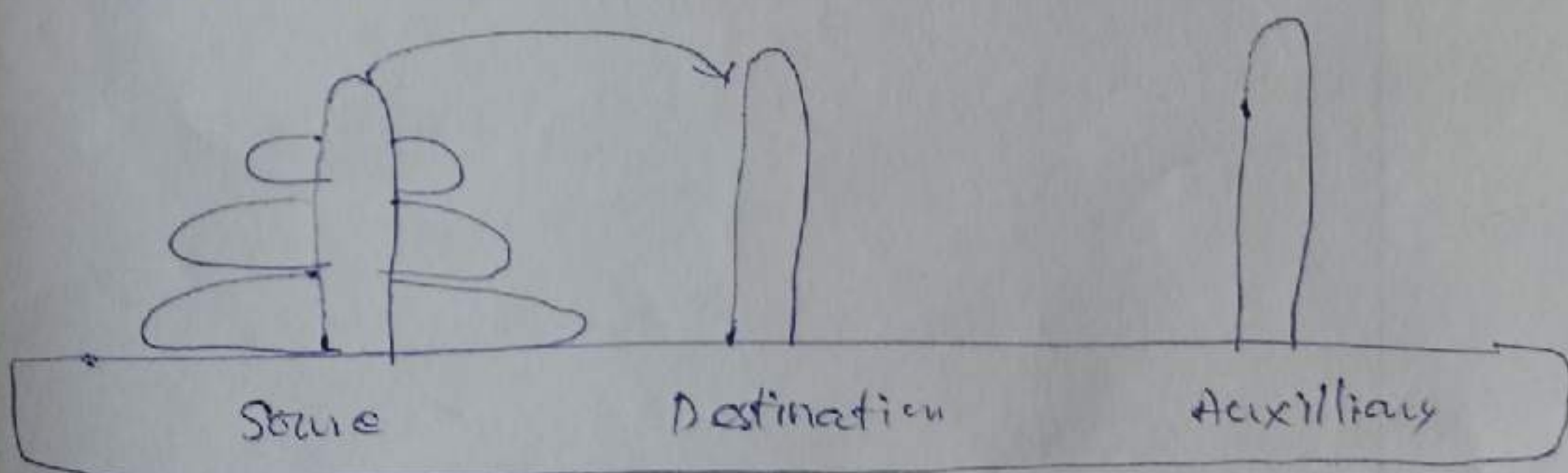
$i = i - 1$

END WHILE

$List[i+1] = X$

1.6.3 Tower of Hanoi

Tower of Hanoi is a mathematical puzzle, which consists of three towers (pegs) and more than one rings, as depicted in figure. These rings are of different sizes and stacked upon in an ascending order, i.e. the smaller one sits over the larger one.



Tower of Hanoi

Algorithm:

Step 1: BEGIN Hanoi (disk, source, dest, aux)
Step 2: IF disk == 1 Then go to step 5 ELSE go to step 4
Step 3: move disk from source to dest AND go to step 7
Step 4: CALL Hanoi (disk-1, source, aux, dest)
Step 5: move disk from source to dest
Step 6: CALL Hanoi (disk-1, aux, dest, source)
Step 7: END Hanoi.

Pseudocode

Procedure Hanoi (disk, source, dest, aux)

IF disk == 1 Then
 move disk from source to dest

ELSE

 Hanoi (disk-1, source, aux, dest)

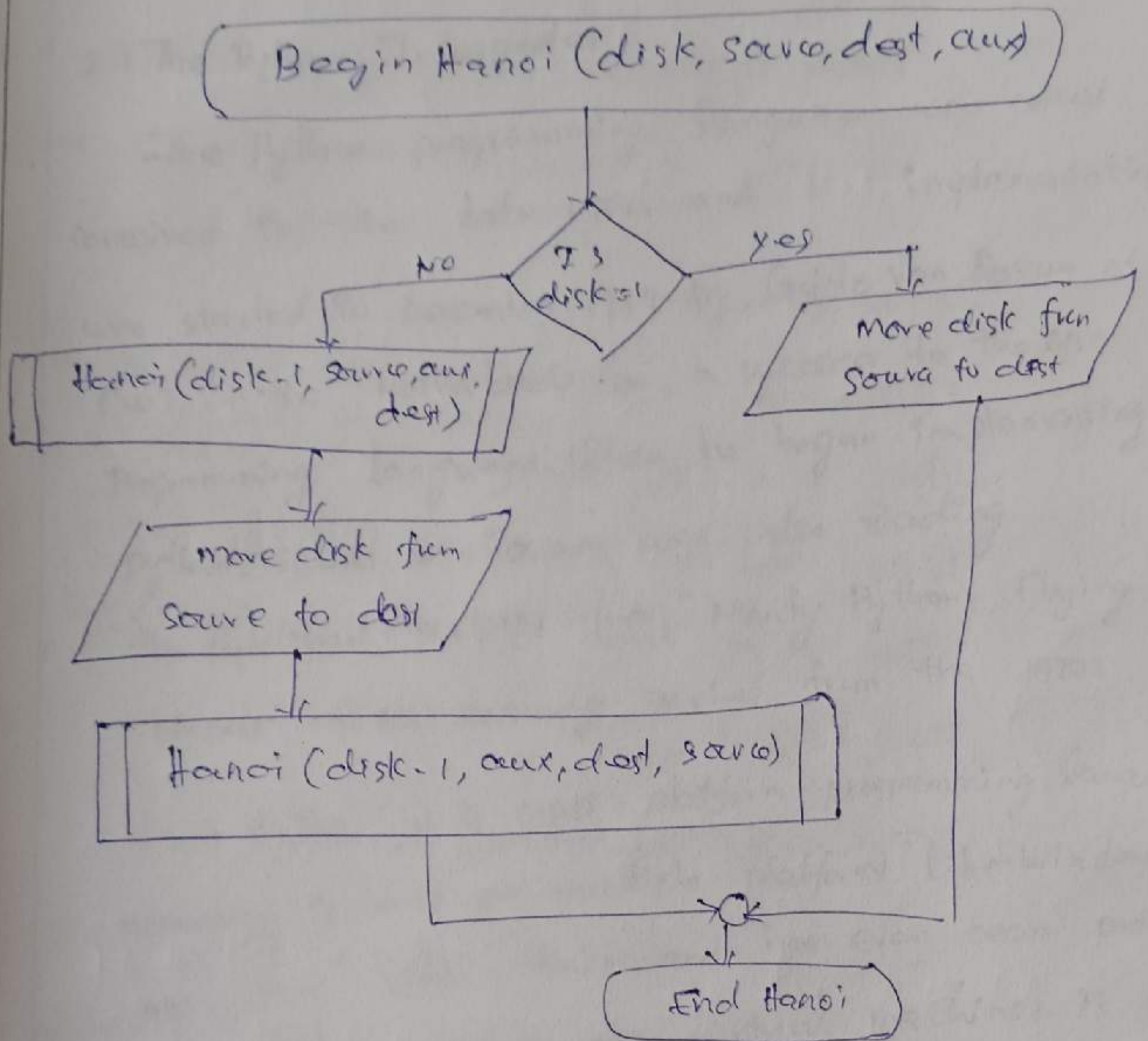
 move disk from source to dest

 Hanoi (disk-1, aux, dest, source)

END IF

END procedure

Flowchart



Unit - II

Data, Expressions, statements

2.1 The Python Interpreter

- The Python programming language was conceived in the late 1980s and its implementation

was started in December 1989 by Guido van Rossum at

CWI in the Netherlands as a successor to the ABC

programming language. When he began implementing

Python, Guido van Rossum was also reading

the published scripts from "Monty Python's Flying

Circus" a BBC comedy series from the 1970s.

(→) Python is a cross-platform programming language, meaning, it runs on multiple platforms like Windows,

Mac OS X, Linux, Unix and has even been ported

to the Java and .NET virtual machines. It is

free and open source.

2.1.1 starting the Interpreter

- The python interpreter is a program that reads and executes Python code.

- After installation, the python interpreter lives in the installed directory.

There are two ways to start python

1) script mode

2) Interactive mode

• Now at the command window, go to the location of this file. You can use the `cd` command to change directory.

• To run the script, type, `python hello_world.py` in the command window. We can see the output as `Hello world !`

2.2. Values and Types

• A value is one of the basic things a program works with, like a letter or a number. Some example values are `5`, `83.0` and `"Hello, world!"`.

• These values belong to different types: `5` is an integer, `83.0` floating point number, and `"Hello world!"` is a string.

2.2.1 Standard Data Types

• Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types.

- Numbers

- String

- List

- Tuple

- Dictionary

2.2.1.1 Python Numbers

Number data types store numeric values. Number objects are created when you assign values to them.

```
var1 = 1
```

```
var2 = 10
```

you can also delete the reference to a number object by using the del statement. The syntax of the del statement is

```
del var1 [, var2 [, var3 [, ... [, varN]]]]
```

you can delete a single object or multiple objects by using the del statement.

```
del var
```

```
del var-a, var-b
```

Python supports a different numerical types

- int (signed integers)

- long (long integers, they can also be represented in octal and hexadecimal)

- float (floating point real values)

- complex (complex numbers)

int	long	float	complex
10	51924361L	0.0	3.14j
1000	-0x19323L	15.20	45j
-786	0122L	-21.7	7.22e-36j

2.2.1.2 Python Strings

Strings in python are identified as a contiguous set of characters represented in the quotation mark. python allows for either pairs of single or double quotes. subsets of strings can be taken using the slice operator (`[]` and `[:]`) with indexes starting at 0 in the beginning of the string and working their way from left to the end.

The plus(+) sign is the string concatenation operator and asterisk(*) is the repetition operator.

```
str = 'python programming'
```

```
print str
```

```
print str[0]
```

```
print str[-1]
```

```
print str[2:5]
```

```
print str[3:]
```

```
print str * 2
```

```
print str + 'course'
```

2.2.1.3 Python Lists

- Lists are the most versatile of python's compound data types. A list contains items separated by commas and enclosed within square brackets (`[]`).

The values stored in a list can be accessed using the slice operator (`[]` and `[:]`) with indexes starting at 0 in the beginning of the list and working their way to end.

```
list = ['Hoi', 123, 1.75, 'vinu', 100, 25]
smallest = ['vinu', 100]
```

Print list

Prints complete list

```
print list[0]
```

Prints first element of the list

```
print list[-1]
```

Prints last element of the list

```
print list[2:]
```

Prints elements starting from 3rd element

```
print smallest*2
```

Prints list two times

2.2.14. Python Boolean

A Boolean type was added to python 2.3

The two new constants were added to the builtin module, True and False. True and False are simply set to integer values, as 1 and 0 and aren't a different type:

```
>>> bool(1)
```

```
True
```

```
>>> bool(0)
```

```
False
```

2.2.2 Data type conversion

Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type name as a function.

Function	Description
<code>int(x[, base])</code>	converts <code>x</code> to an integer, <code>base</code> specifies the base if <code>x</code> is a string.
<code>long(x[, base])</code>	converts <code>x</code> to a long integer, <code>base</code> specifies the base if <code>x</code> is a string.
<code>float(x)</code>	convert <code>x</code> to a floating-point number.
<code>str(x)</code>	Converts object <code>x</code> to a string representation.
<code>repr(x)</code>	Converts object <code>x</code> to an expression string.
<code>list(s)</code>	converts <code>s</code> to a list.
<code>chr(i)</code>	converts an integer to a character.
<code>unichar(x)</code>	converts an integer to a unicode character.

2.3. Variables

A variable is a name that refers to a value. Variable reserved memory locations to store values. This means that when you create a variable you reserve some space in memory. Based on the data type also a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.

2.3.1 Assignment Statements:

- An assignment statement creates a new variable and gives it a value.

```
>>> message = 'Introducing Python Variable'
```

```
>>> num = 15
```

```
>>> radius = 5.4
```

- Here the first assigns a string to a new variable named message, the second gives the integer 15 to num; the third assigns floating point value 5.4 to variable radius.

1. The first part of the paper

is a description of the various kinds of
the various kinds of the various kinds of
the various kinds of the various kinds of

the various kinds of the various kinds of
the various kinds of the various kinds of
the various kinds of the various kinds of

The remainder of the paper is devoted
to the various kinds of the various kinds of

the various kinds of the various kinds of

the various kinds of the various kinds of

the various kinds of the various kinds of

the various kinds of the various kinds of

the various kinds of the various kinds of

There are some reserved (key words)
which are used as a shorthand

the various kinds of the various kinds of
the various kinds of the various kinds of
the various kinds of the various kinds of

2.4. Expressions and Statements

An expression is a combination of values, variable, and operators. A value all by itself

is considered an expression and so is a variable, so the following are all legal expressions.

`>>> 50`

`>>> 10 < 5`

`False`

`>>> 50 + 20`

`70`

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

`>>> x = 25`

`>>> print(x)`

Difference between a statement and an expression

A statement is a complete line of code that performs some action, while an expression is any section of the code that evaluates to a value.

Expressions can be combined "horizontally"

into larger expressions using operators, while

statements can only be combined "vertically"

by writing one after another, or with block

constructs. Every expression can be used as

a statement, but most statements cannot

be used as an expression.

2.5 Operators

- Operators are special symbols in python that carry out computation. The value that the operator operates on is called the operand.

Python Operators

- Arithmetic operators

- Comparison (Relational) operators

- Logical (Boolean) operators

- Bitwise operators

- Assignment operators

- Special operators

2.5.1 Arithmetic Operators

- Arithmetic operators are used to perform

mathematical operations like addition, subtraction, multiplication,

Operator	Meaning	Example
+	Add two operands or unary plus	$x + y$ $+ 2$
-	Subtract right operand from the left or unary minus	$x - y$ $- 2$
*	Multiply two operands	$x * y$
/	Divide left operand by the right one	x / y
%	Modulus - remainder of the division of left operand by the right	$x \% y$

2.5.2 Comparison or Relational Operators

- Comparison operators are used to compare values. It either returns True or False

Operator	Meaning	Example
>	Greater than - True if left operand is greater than the right	$x > y$
<	Less than - True if left operand is less than the right	$x < y$

$==$ Equal to - True if both operands are equal $x == y$

$!=$ Not equal to - True if operands are not equal $x != y$

$>$ Greater than or equal to - True if left operand is greater than or equal to the right $x > y$

$<$ Less than or equal to - True if left operand is less than or equal to the right $x < y$

2.5.3 Logical Operators

- Logical operators are the and, or, not

Operator	Meaning	Example
and	True if both the operands are true	$x \text{ and } y$
or	True if either of the operands is true	$x \text{ or } y$
not	True if operand is false	not x

2.5.3 Bitwise Operators

- Bitwise operators act on operands as if they were string of binary digits. If operator bit by bit, hence the name.

Example

operator Meaning

& Bitwise AND $x \& y = 0$ (0000 0000)

| Bitwise OR $x | y = 14$ (0000 1110)

~ Bitwise NOT $\sim x = -11$ (1111 0101)

^ Bitwise XOR $x \wedge y = 14$ (0000 1110)

>> Bitwise right shift $x >> 2 = 2$ (0000 0010)

<< Bitwise left shift $x << 2 = 40$ (0010 1000)

2.5.5 Assignment Operators

Assignment operators are used in python to assign values to variables.

Operator Example Equivalent to

= $x = 5$ $x = 5$

+= $x += 5$ $x = x + 5$

-= $x -= 5$ $x = x - 5$

*= $x *= 5$ $x = x * 5$

/= $x /= 5$ $x = x / 5$

//= $x //= 5$ $x = x // 5$

2.5.6. Special Operators

Python language offers some special type of operators like the identity operator or the membership operator. They are described below with examples.

2.5.6.1 Identity Operators

`is` and `is not` are the identity operators in python. They are used to check if two values (or variables) are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

Operator	Meaning	Example
<code>is</code>	True if the operands are identical	<code>x is True</code>
<code>is not</code>	True if the operands are not identical	<code>x is not True</code>

2.5.6.2 Membership Operators

`in` and `not in` are the membership operators.

Operator	Meaning	Example
<code>in</code>	True if value/variable is found in the sequence	<code>5 in x</code>
<code>not in</code>	True if value/variable is not found in the sequence	<code>5 not in x</code>

2.6. Precedence of python Operators

The combination of values, variables, operators and function called **expression** if it contains more than one operator, the order of evaluation depends on the precedence of operators.

Operators	Meaning
-----------	---------

()

parentheses

**

Exponent

+x, -x, ~x

Unary plus, Unary minus,

Bitwise Not

*, /, //, %

Multiplication, Division, Floor division, modulus

+, -

Addition, subtraction

<<, >>

Bitwise shift operators

&

Bitwise AND

^

Bitwise XOR

|

Bitwise OR

==, !=, >, <, <=, is, isnot, in, not in

Comparison, Identity,

Membership operators.

2.7 Associativity of python Operators

- Associativity is the order in which an expression is evaluated that has multiple operators of the same precedence. Almost all the operators have left-to-right associativity.

```
>>> 10 * 7 // 3
```

```
23
```

```
>>> 10 * (7 // 3)
```

```
20
```

```
>>> (10 * 7) // 3
```

2.7.1 Non Associative Operators

- Some operators like assignment operators and comparison operators do not have

associativity in python. There are separate rules for sequences of this kind of operators and cannot be expressed as associativity.

Example

$x < y < z$ neither means $(x < y) < z$ nor

$x < (y < z)$, $x < y < z$ is equivalent to $x < y$ and

$y < z$, and is evaluated from left-to-right.

2.8 Tuple Assignment

It is often useful to swap the values of two variables. With convention of assignment, you have to use a temporary variable. For example, to swap a and b:

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

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

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

- This solution is cumbersome, tuple assignment is more elegant:

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

- The left side is a tuple of variables, the right side is a tuple of expressions. Each value is assigned to its respective variable.

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

ValueError: too many values to unpack

```
>>> addr = 'monty@python.org'
```

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

The return value from `split` is a list with two elements; the first element is assigned to `uname`, the second to `domain`.

2.9 Comments

As programs get bigger and more complicated, they get more difficult to read. Formal languages are dense, and it is often difficult to look at a piece of code and figure out what it is doing, or why. For this reason, it is a good idea to add notes to your programs to explain in natural language what the program is doing. These notes are called comments and they start with the # symbol.

compute Area of a triangle using Base and Height
$$\text{area} = (\text{base} * \text{height}) / 2$$
 # Area of a triangle using Base and Height

- If we have comments that extend multiple lines, one way of doing it is to use hash (#) at the beginning of each line

This is a long comment
and it extends

to multiple lines.

- Another way is by using triple quotes, either ''' or """

2.9.1 Docstring in Python

Docstring is short for documentation string.

It is a string that occurs as the first statement in a module, function, class or method definition. We must write what a function/class does in a docstring. Triple quotes are used while writing docstrings.

```
def area(r):
```

```
    """Compute the area of circle"""  
    return 3.14159 * r ** 2
```

2.10 Modules and Functions

2.10.1 Functions

Functions are an important step to cover, when introducing additional complexity into your programming. If a variable is a named container for data, then a function is a named container for a block of code that can be executed on demand. This is useful if you have a program that executes a certain operation lots of times.

- A function is a named container for a block of code. There are two types of functions: the ones that user-defined functions and library functions / PREDEFINED function, the ones that are included in python natively, which carry out common procedures, such as converting an integer to a string, or finding the length of a string.

(i) User-defined function

- Rules for user-defined function

- Use keyword as 'def'

- The functions name to be followed by 'def'

- The pair of parentheses () is appear next to functions name.

- The arguments are appear within parentheses.

- The functions header should be terminated by colon (:)

- Code the statements of functions block should be indent

- 'return' statement is optional.

Syntax:

- def function name (arguments):

statement 1

statement 2

statement n

return

(ii) Built-in (pre-defined) functions

- We have learned how to make our own functions, and how to use them, but there

are some operations that are carried out

often enough that python has included

them for use in any program. The way you

call them is exactly the same as calling

your own, but you don't have to define

them first.

str()

- The string conversion function. There are many times in scripting when you have

a variable that contains a number, or some other type of data, but you need to convert

that value to a string in order to do

something with it, normally to print it out onto the screen.

The `str()` function finds the closest string match it can find for the value of 'true' and returns it, meaning that we can output it nicely.

`len()`

- To find length of the string.

`int()`

- The built-in function to convert to an integer is `int()`. This function will return the integer conversion of a non integer number.

`range()`

`Range` is a function that you use to create a list of numbers following a certain pattern.

2.10.2: Modules

- Modules are quite easy to create. They are simply python files, like your regular scripts.

To create a module, write one or more functions in a text file, then save it with a .py extension.

Types

(i) User-defined module

(ii) Pre-defined module

- Create a new file in IDE or text editor and create a function for calculating tax on a product.

1. `def addTax (price, tax):`

2. `new price = price / 100 * (100 + tax)`

3. `return new price.`

If we save this file with a .py extension

in the same directory as our other script,

we can use it as a module! It's important

to give it a descriptive name so you know what it does when you come back.

(i) Importing modules - User defined module

- To use modules, we can ^{use} either import or from keyword. import is the simplest and most

common, so let's we have to specify the name of the module, which is simply the filename, without the .py extension.

Cii) Built-in Modules - pre defined module

- There are plenty of built in modules just as there are with built in functions.

This is where python really excels. python has such an extensive range of built-in modules, there's no way to cover them all.

* Math

* random

* os

* datetime

random:

- It is used to generate random numbers, for example when using python to create a website, they can be used in making password database more secure by powering a random page feature.

math:

- The math module provides access to mathematical constants and functions.

data time

- If you need to work with dates and times, then the data time module is your friend. For making an website with comments, or a blog, then we have to display the age, or the time they are created.

Flow of Execution

- Execution always begins at the first statement of the program. Statements are executed one at a time, in order, from top to bottom. Function definitions do not alter the flow of execution of the program, but remember that statements inside the function are not executed until the function is called.

- Function calls are like a detour in the flow of execution. Instead of going to the next statement, the flow jumps to the first line of the called function, executes all the statements there, and then comes back to pick up where it left off.

Parameters and Arguments

- Some of the functions we have seen that it requires arguments. For example, when call `math.sin`, it pass a number as an argument. Some function take more than one argument: `math.pow` takes two, the base and the exponent. Inside the function, the arguments are assigned to variables called parameters.

```
def print_twice (flower):  
    print (flower)  
    print (flower)
```

- This function assigns the argument to a parameter named `twice`. When the function is called, it prints the value of the parameter (whatever it is) twice.

```
>>> print_twice ('world')
```

world

world

```
>>> print_twice (56)
```

56

56

Arguments

- The term arguments also referred as parameters in a function.

- Arguments are used to pass certain values to execute specific block of code.

- It act as temporary variable for a function.

Types:

- Required arguments

- Default arguments

- Keyword arguments

- variable-length arguments

(i) Required arguments

- It is a arguments to pass values to

the function in Correct Positional Order.

Ex:

```
def add(a,b)
```

```
    c=a+b
```

```
    print(c)
```

```
add(10,20)
```

```
add(15)
```

- last line shows error because it has

only one arguments.

Unit-III - Control Flow, Functions

i) Decision Making (Conditionals)

- python language provides 4 general categories of control structures.

(i) Sequential structure

↳ In which instructions are executed

in sequence

ex: $i = i + 1$

$j = j + 1$

The above statements are executed one by one.

(ii) Selection structure

↳ Here the sequence of instructions

is determined by using the result of the condition

ex: $\text{if } (x > y):$

$i = i + 1$

else:

$j = j + 1$

- If the condition is true, then the statement $i = i + 1$ will be executed otherwise it executes $j = j + 1$

(iii) Iteration structure

- In which statements are repeatedly executed. These forms program loops.

ex: `for i in range(5):`
`print(i)`

Here the statement `print(i)` will be executed 5 times and value of `i` will change from 0, 1, 2, 3, 4.

(iv) Encapsulation structure

↳ In which the other compound structures are included.

python language provides the following conditional statements

if statement

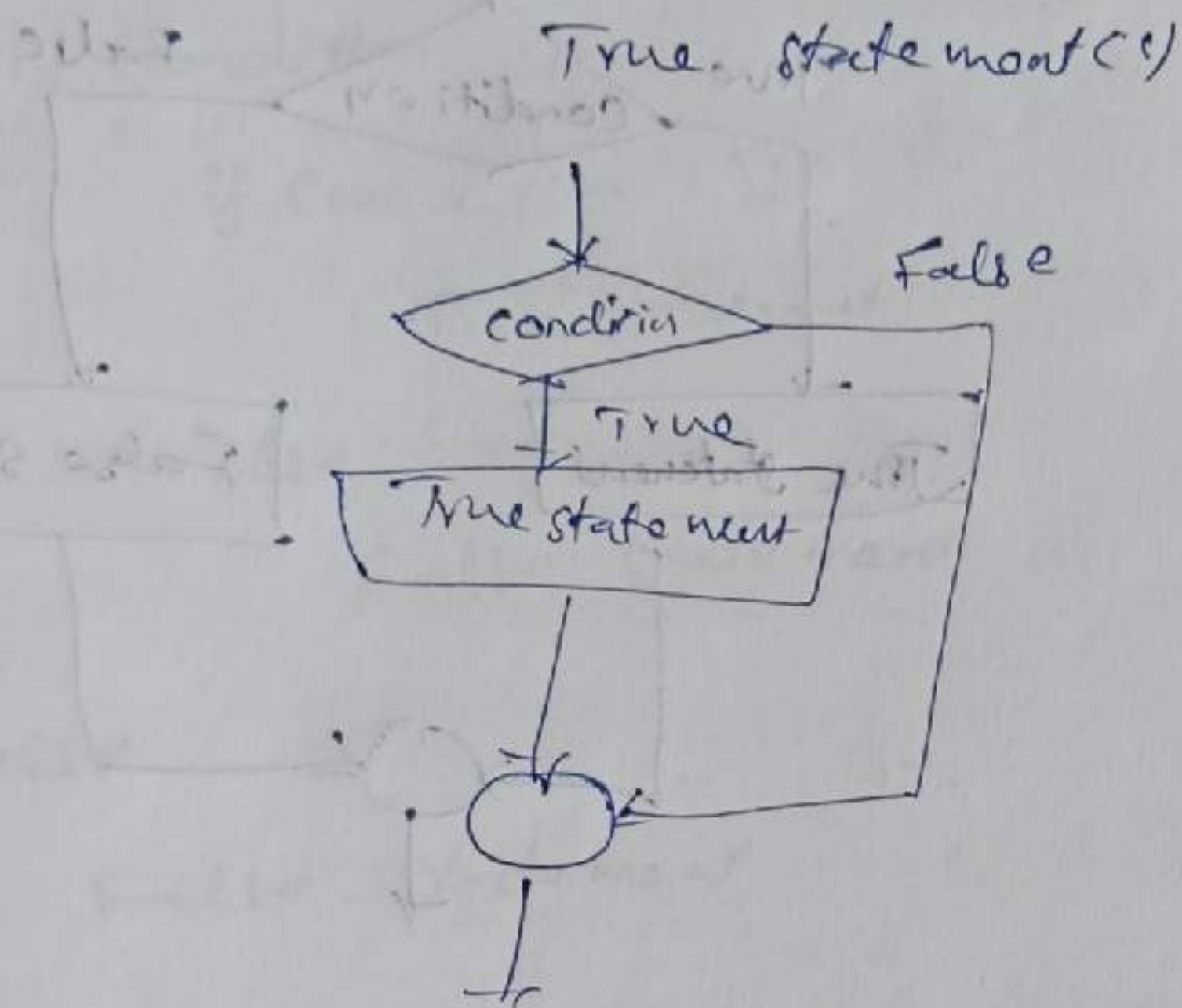
if... else statement

nested if... else

if... else ladder.

1.2. The if statement

Syntax: if (condition is true):



EX: `if (input < 10):`

`i = int(input("Enter the number < 10: "))`

`if (i < 10):`
 `print("The entered number is < 10", i)`

Output:

Enter the number < 10:

The entered number is < 10 5

1.3. The if-else statement

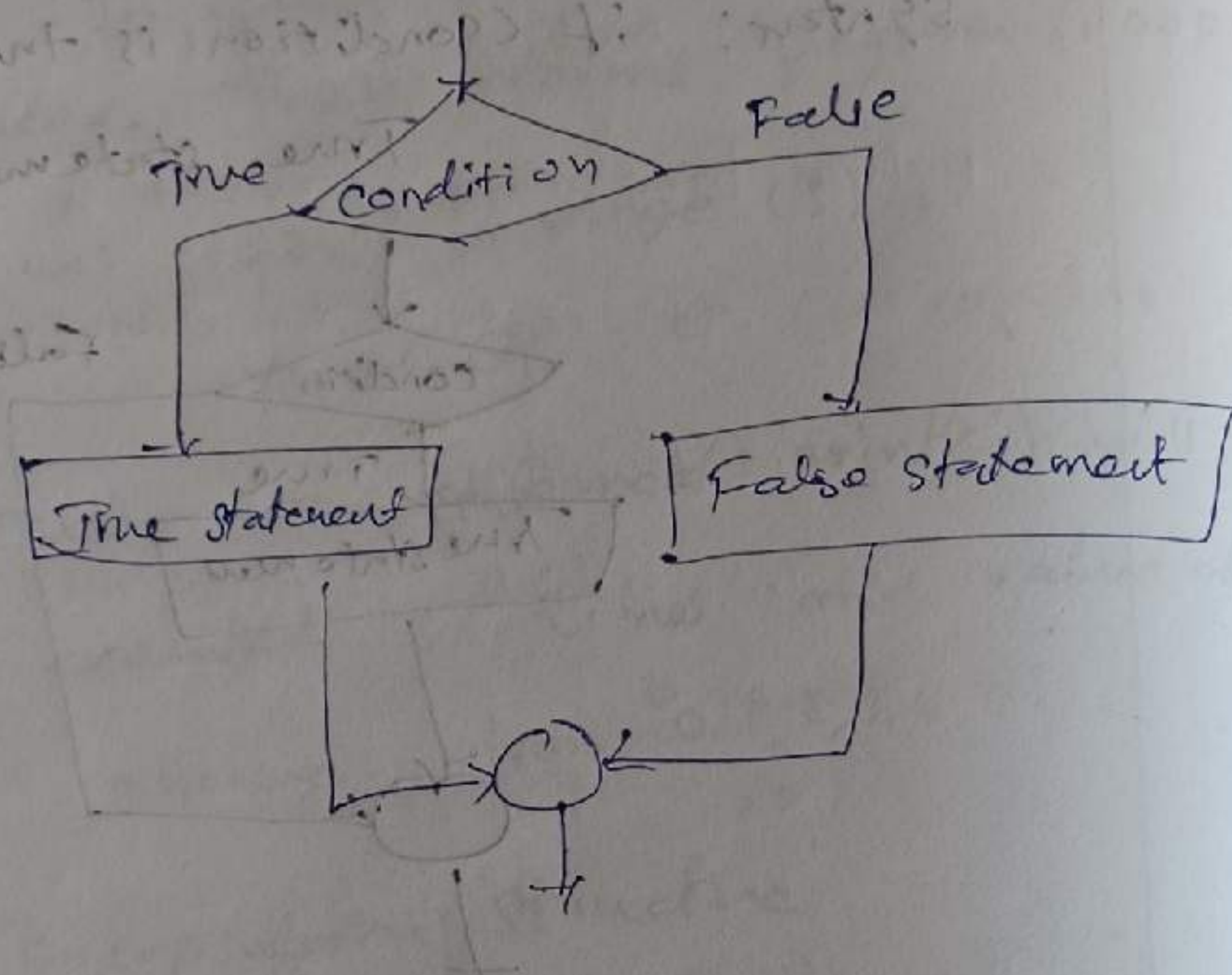
Syntax:

if (condition):

True statements

else:

False statements



Ex:-

```

? = int(input("enter the number:"))

```

```

if (x < 10):

```

```

    print("The no is <")

```

```

else

```

```

    print("The no is ", i)

```

Out put:-

enter the number: 5

The no is 5

1.4 Nested if... else statement

Syntax:

if (condition 1):

if (condition 2):

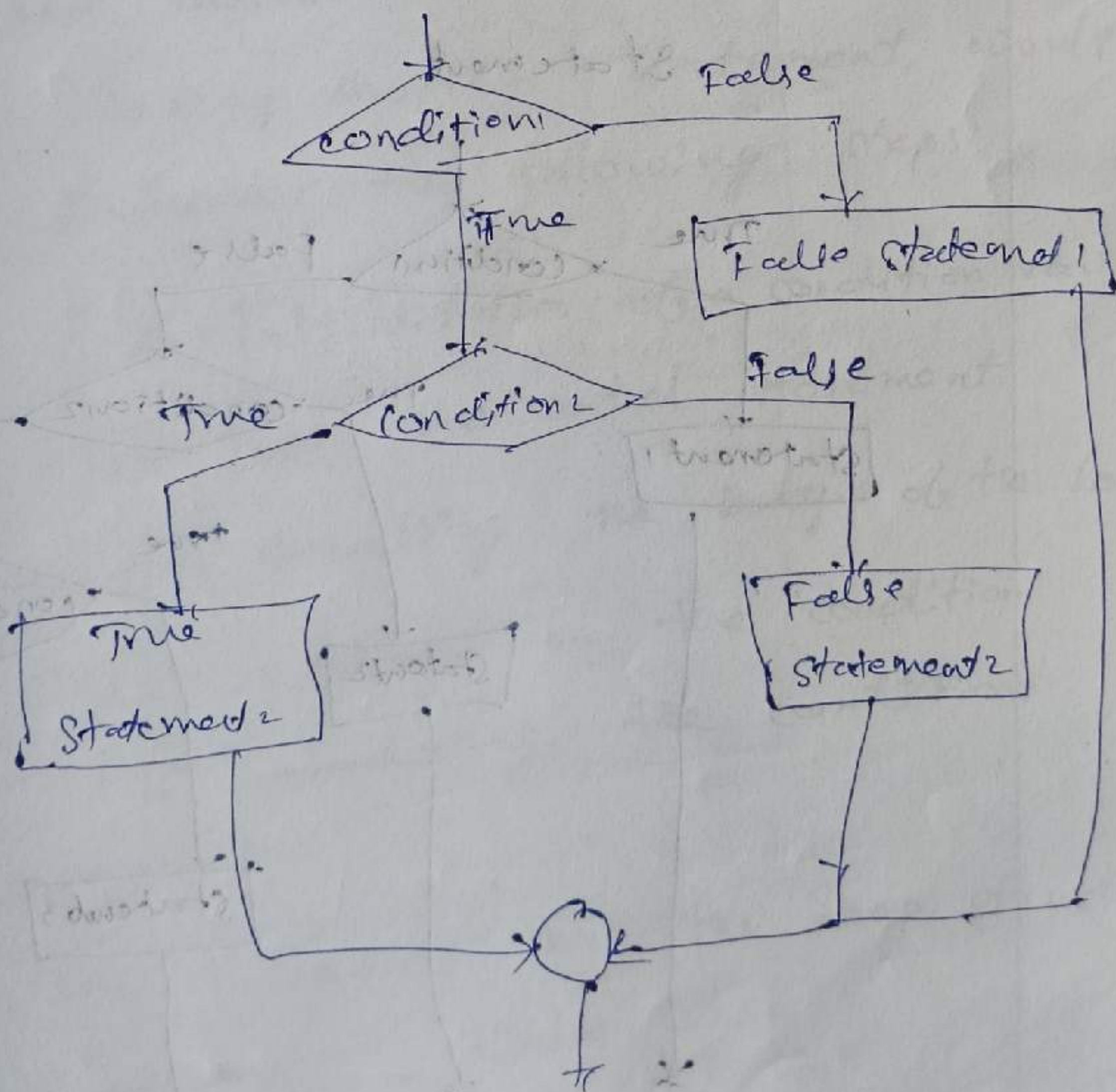
True statement

else:

False statement

else

False statement



1.5 The if... else statement

Syntax:

if expression:

statement

elif expression 2:

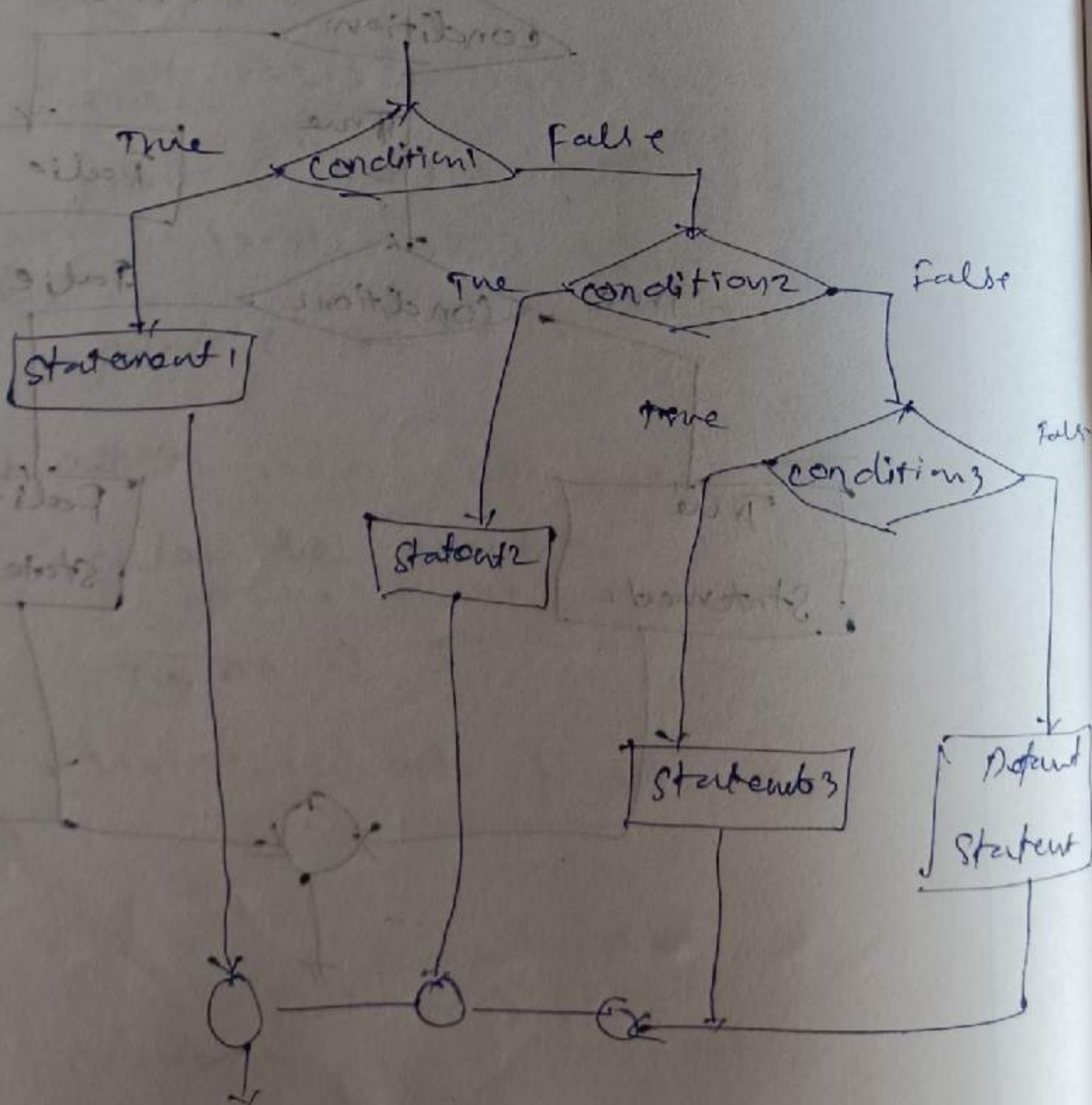
statement 2

elif expression 3:

statement

else:

statement



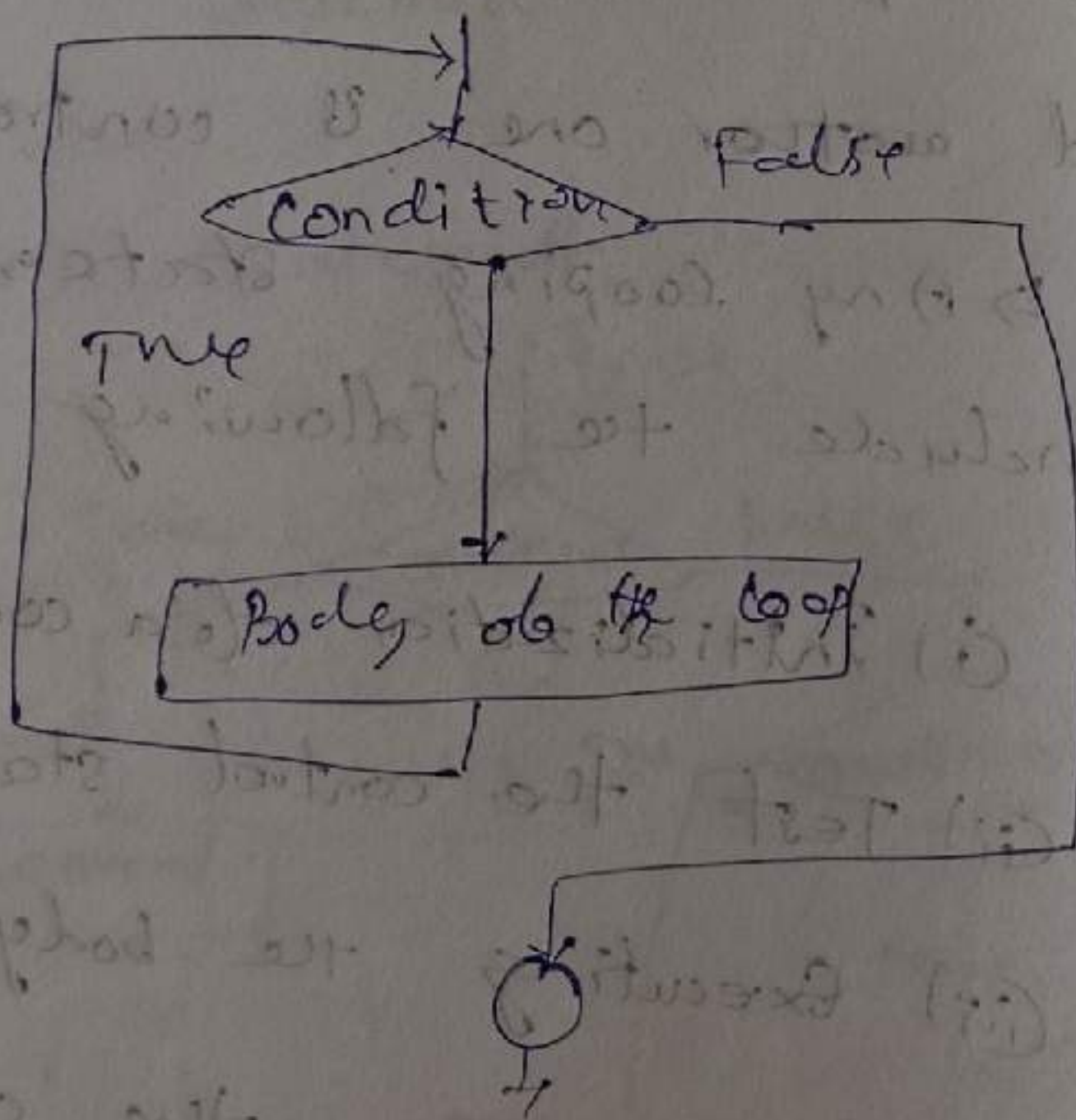
The while loop

→ The while loop is an entry-controlled loop statement, which means the condition is evaluated first and if it is true, then the body of the loop is executed.

Syntax:

while (condition):

body of the loop



Example:

i = 1; Sum = 0

while (i <= 10):

Sum = Sum + i;

i = i + 1

print ('Sum of no upto 10 is', Sum)

Output:

The Sum of numbers upto 10 is 55

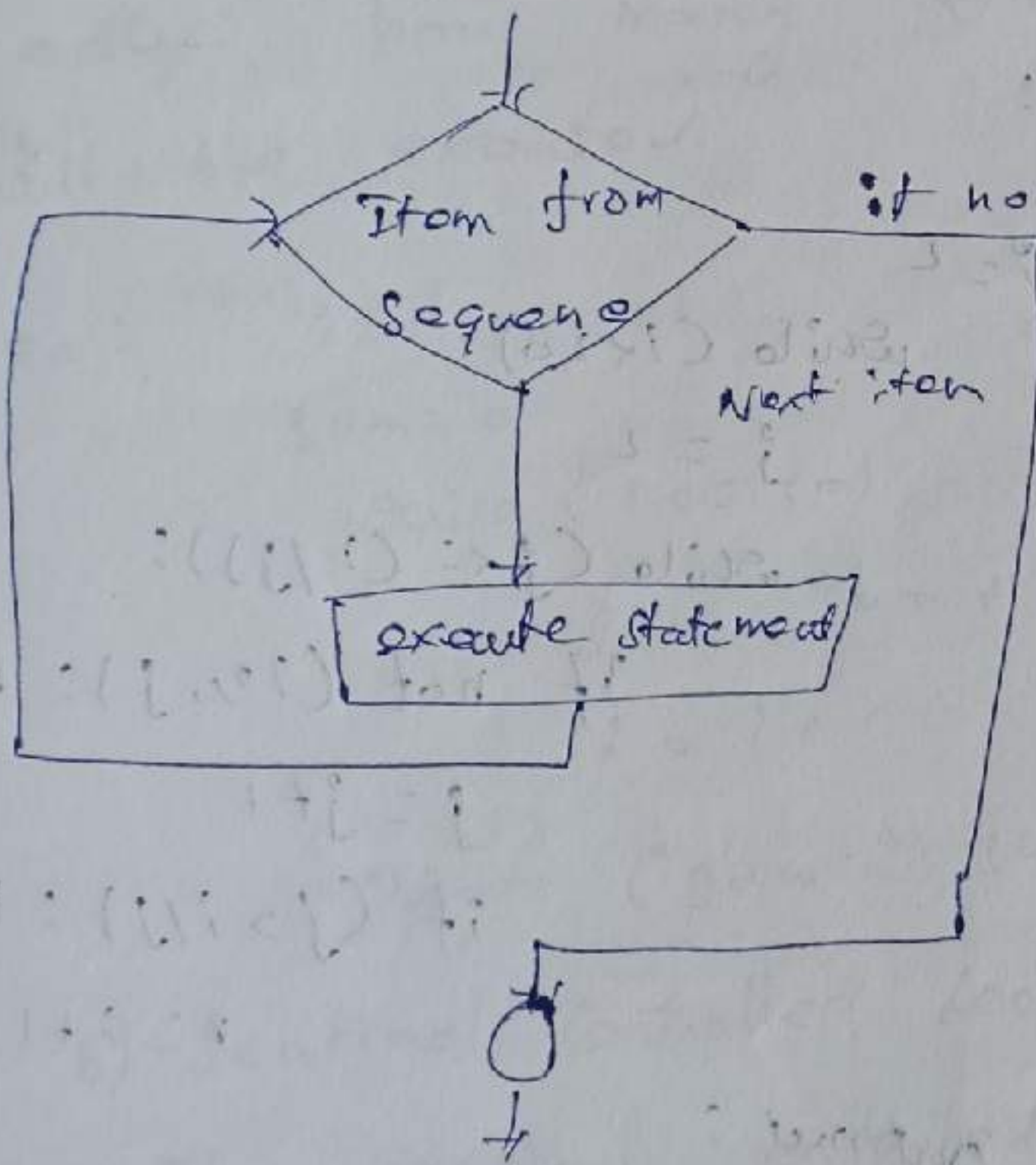
The for loop

The for loop is used to execute a set of instructions repeatedly, until the condition becomes false.

Syntax:

```
for iterating_var in sequence:
```

```
    statements
```



Ex:

```
for word in 'muni':  
    print ('char in word:', word)
```

Output:

```
char in word: m  
char in word: u  
char in word: n  
char in word: i
```

python nested loops :

- The loop within the loop is called nested loop.

Syntax:

while expression:

while expression:

statement

statement

ex:

```
i = 2
while (i < 20):
    j = 2
    while (j <= (i / j)):
        if not (i % j): break
    j = j + 1
    if (j > i / j): print(i, "is prime")
    i = i + 1
```

Output :

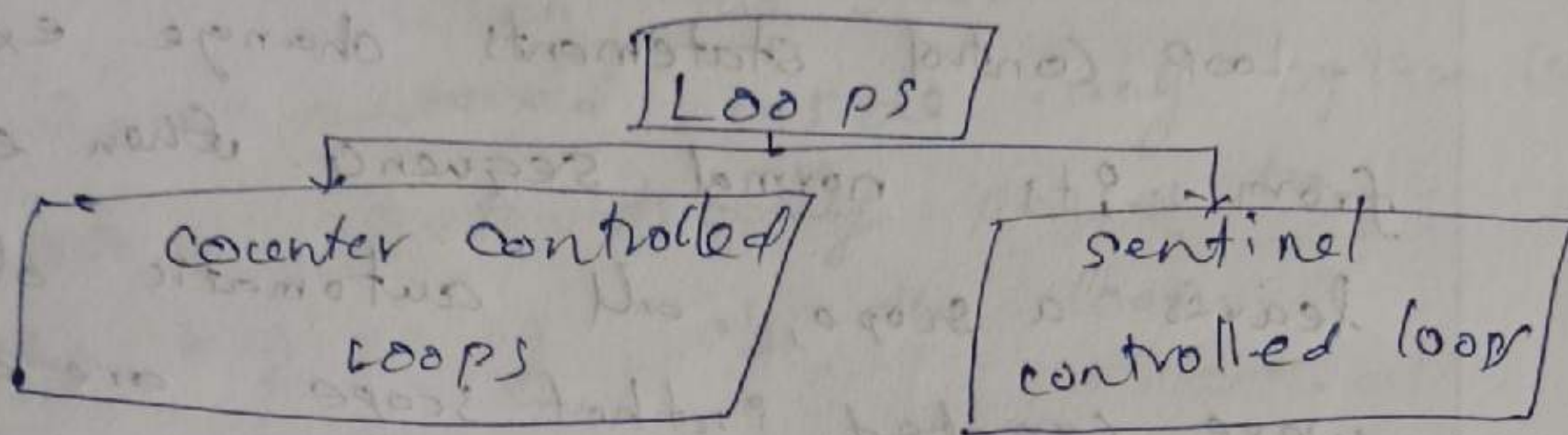
2 is prime

3 is prime

5 is prime

7 is prime

Types of loops



a) Counter controlled loops:

The loop are used in a situation when the programmer already knows exactly how many times the loop will be executed.

ex: `for i = 1 to 10`

`sum = 0`

`while (i <= 10)`

`sum = sum + i`

`i = i + 1`

`Print "sum:", sum`

b) Sentinel controlled loop:

The sentinel controlled loops are used in a situation, when the programmer does not know already the exact number of times the loop will be executed.

`sum = 0; i = 0`

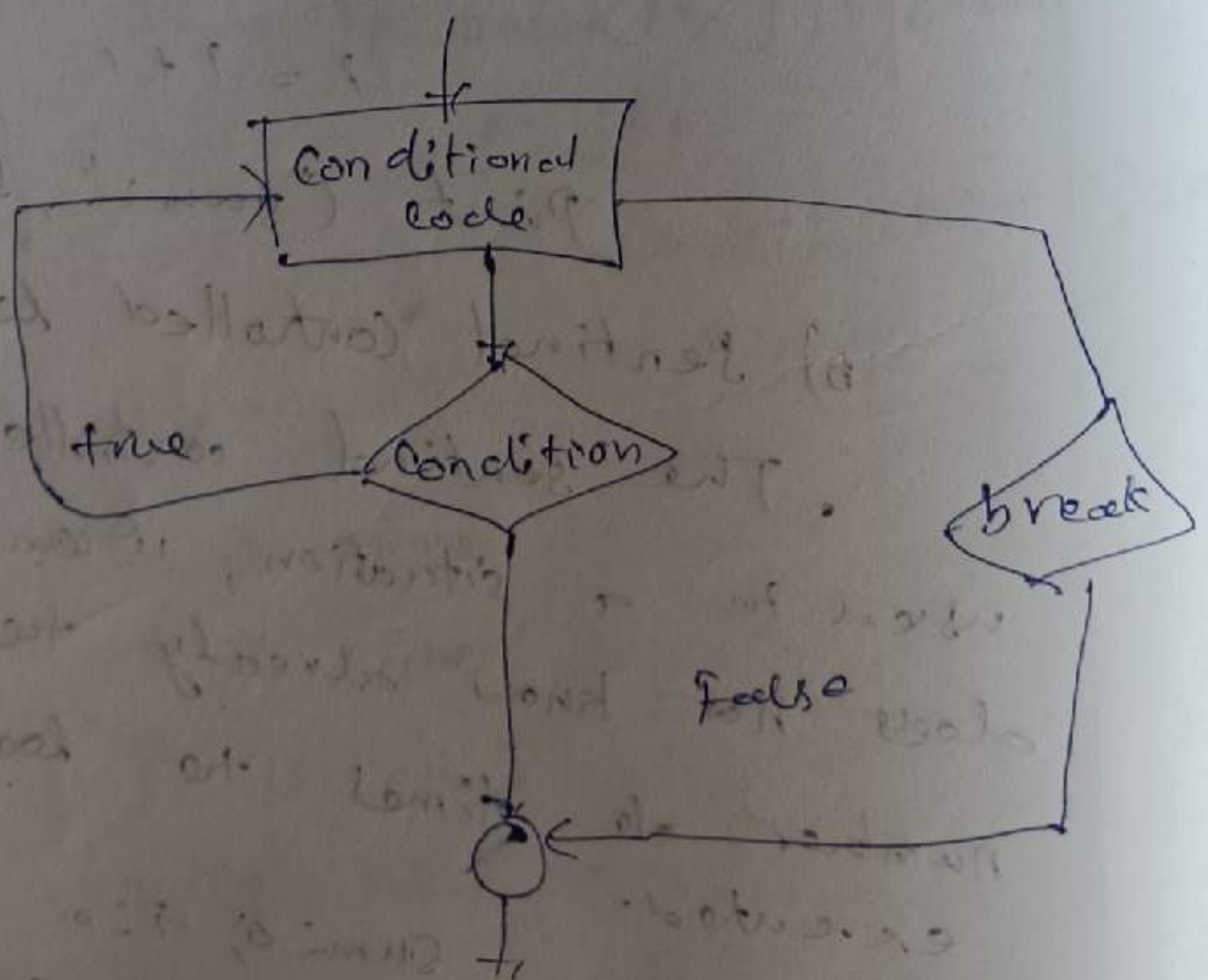
Loop Control Statements

- Loop Control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

The break statement

The break statement is used to terminate the loop. When the keyword break is used inside any python loop, control is automatically transferred to the first statement after the loop.

Syntax: `break`

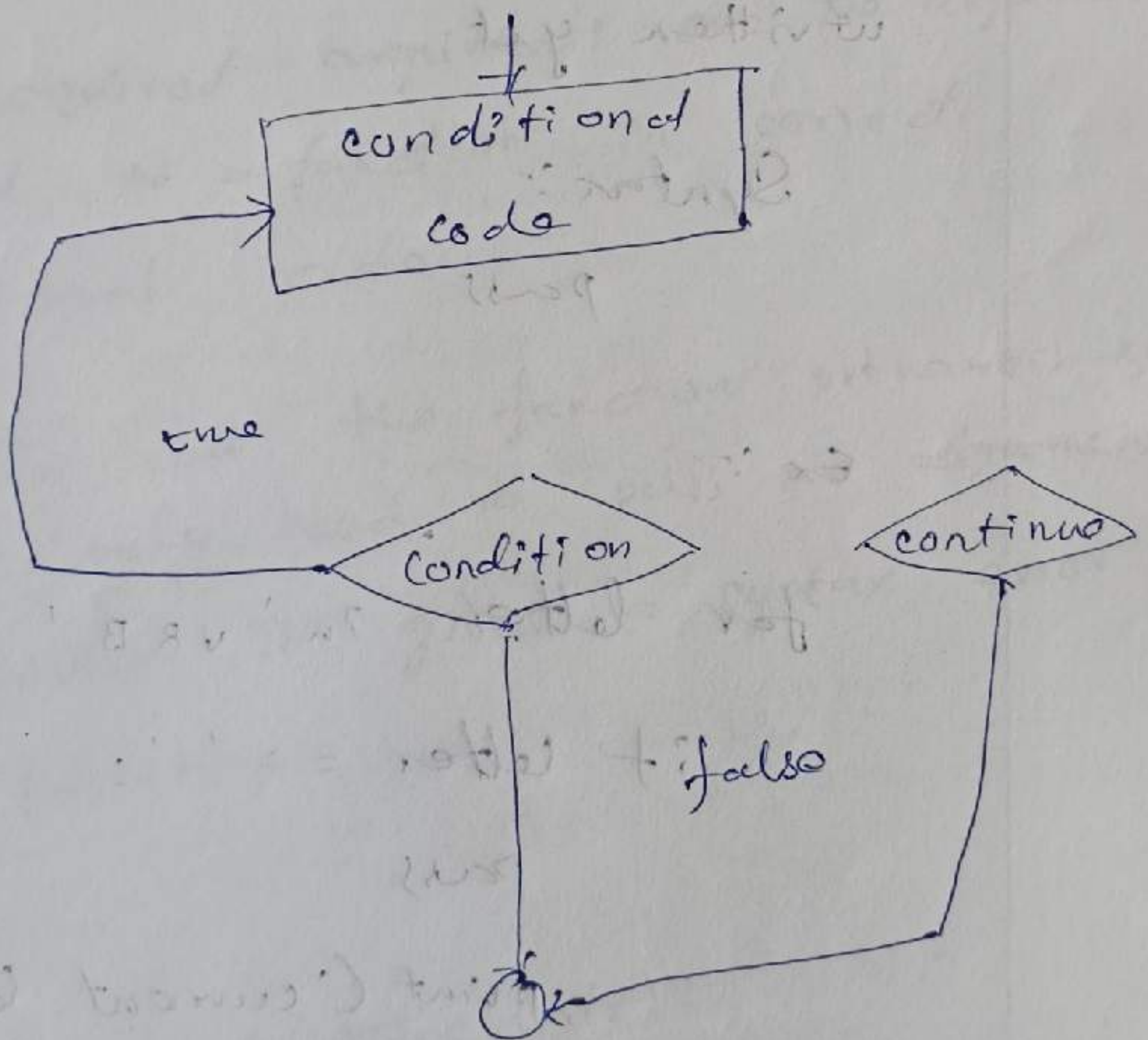


python continue statement

When the statement continue is encountered inside any python loop control automatically passes to the beginning of the loop.

Syntax:

Continue



Like break statement, the continue statement also associate with if statement.

python pass Statement

The pass statement is a null operation, nothing happens when it executes. The pass is also useful in places where your code will eventually go, but has not been written yet.

Syntax :-

pass

Ex :-

```
for letter in 'VRB':
```

```
    if letter == ' ':
```

```
        pass
```

```
    print('current letter:', letter)
```

output

current letter: V

current letter: R

current letter: B

Function arguments

- * Required arguments
- * Keyword arguments
- * Default arguments
- * Variable-length arguments.

Required arguments

- Required arguments are the arguments passed to a function in correct positional order.

- To call the function `prtname()`, you definitely need to pass the argument, otherwise it gives a syntax error as follows

ex:

```
def prtname(str):
```

```
    print(str)
```

```
    return;
```

```
prtname()
```

Keyword arguments

- Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

ex:

```
def putname(str):
```

```
    "This prints string"
```

```
    print(str)
```

```
    return;
```

```
putname(str = 'VRB')
```

Output :

VRB publishers.

- This function `putname(str)` is defined with one argument, and while calling that function we need to pass an argument to that function.

Default arguments

- A default argument is an argument that assumes a default value, if a value is not provided on the function call for that argument.

- The following example gives an idea on default arguments, it prints default phone number if it is not passed.

```
ex:
def printInfo (name, phno = '123'):
    print ("Name:", name)
    print ("Phone no:", phno)
```

```
    return;
```

```
printInfo (name = 'VAR')
```

```
printInfo (name = 'VAR', phno = '1234')
```

Output:

Name: VAR

phno: 123

Name: VAR

phno: 1234

Variable Length arguments:

- When we need to process a function for more arguments than you specified while defining the function. These arguments are called variable-length arguments and are not named in the function definition, unlike required and default arguments.

Syntax:

```
def function_name([formal_args])*var_arg:
```

```
    "function docstring"
```

```
    function suite
```

```
    return [expression]
```

- An asterisk (*) is placed before the variable name that holds the values of all non-keyword variables arguments.

- This tuple remains empty if no additional arguments are specified during the function call.

Unit V: Files, Modules, Packages

Python files

File Types

1) Text files are structured as a sequence of lines, where each line includes a sequence of characters.

2) Binary files of any type of file that is not a text file. Binary files can only be processed by an application that knows or understands the file's structure.

The open function

Syntax:

file object = open(file_name [access mode] [buffering])

File opening modes

- r, rb, rt, rbt, w, wb, wt, wbt
a, ab, at, abt

The file object Attributes

- Once a file is opened and you have one file object, you can get various information related to that file

- `file.closed`, `file.mode`, `file.name`,
`file.softspace`

The file `close()` method

- The `close()` method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.

Syntax:

`fileObject.close()`

Reading and writing files

The write() method

- The write() method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.

Syntax:

fileObjed. write(string)

Ex:-

```
f1 = open("vib.txt", "w")
```

```
str = "P S P P"
```

```
f1.write(str)
```

```
f1.close()
```

The read() method

- The read() method reads string from an open file. It is important to note that Python strings can have binary data apart from text.

Syntax:

file Object. read([count])

Ex:

```
f1 = open("vrb. text", "rt")
```

```
str1 = f1.read(10)
```

```
str2 = f1.read(20)
```

```
str3 = f1.read(30)
```

```
print("Read: ", str1)
```

```
print("Read: ", str2)
```

```
print("Read: ", str3)
```

```
f1.close()
```

Output:

Read 10 : problem so

Read 20 : living and psychology

Read 30 : Grammar

Searching through a file

When we are searching through data in a file, it is a very common pattern to read through a file, ignoring most of the lines and only processing lines which meet a particular condition.

ex:

```
fi = open('vrb.txt')
```

```
count = 0
```

```
for line in fi:
```

```
    if line.startswith('problem'):
```

```
        print(line)
```

Output:

problem Solving and python
programming

The seek() and tell() methods

- To know the position of the file pointer, we can use the tell() method. It returns the current position

of the file pointer from the beginning
of the file.

$n = f.tell()$

- In case, we want to move the file
pointer to another position, we can
use the `seek()` method.

`f.seek(Offset, fromwhere)`

Renaming and deleting files

- The `rename()` method takes two
arguments, the current filename and
the new filename

Syntax

`os.rename(current_file_name, new_file_name)`

import os

`os.rename("wh.txt", "test.txt")`

The remove method

we can use the remove method to delete files by supplying the name of the file to be deleted as the argument

Syntax:

```
os.remove(file_name)
```

Ex :-

```
import os
```

```
os.remove("test.txt")
```