UNIT - II

Python provides a control structure which transfer the control from one part of the program to some other part of the program. A control structure is a statement that determines the control flow of the set of instructions.
There are different types of control statements supported by Python like decision control, loop control and jump statement -

Concept of indentation:-

One of the most distinctive features of Python is its use of ~~indentation~~ indentation to mark a block of code. In Python, code blocks are identified by identation rather than using a symbol like curly braces. Without extra symbols, programs are easier to read.
Identation refers to the spaces that are used at the beginning of the statement. The statement with the same indentation belongs to the same group called a suite.

```
>>> if x==1:
>>>       print("hello")
>>>       print('world').
>>> print("end of the program").
```

→ **Decision Control Statement / Conditional Statement.**

These are also known as Selection Control Statements. The decision control statement alters the normal sequential execution of the statement of the program depending on the test condition to be carried out at a particular point in the program. The decision to be taken regarding where the control should transfer depends on the outcome of the test conditions.

* **The if Statement :**

The if statement is used to execute one or more statements only if the condition is true. The syntax is :

```
if condition;
    statement
```

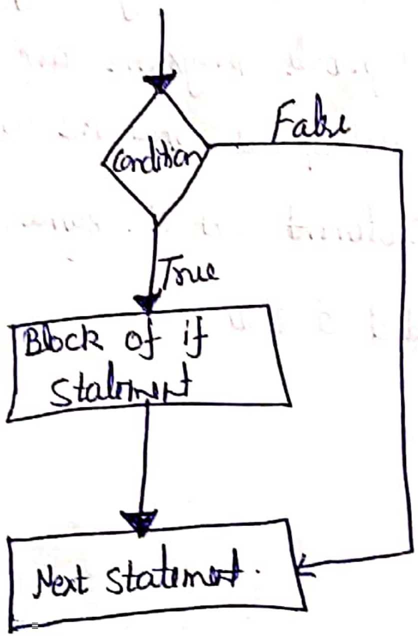


Fig:- flowchart of a simple if statement.

// Program to find a number is even

```
n= int (input ("Enter the number"))
if n%2=0:
    print ("even no")
print ("end of the program")
```

* **The if -- else Statement :-**

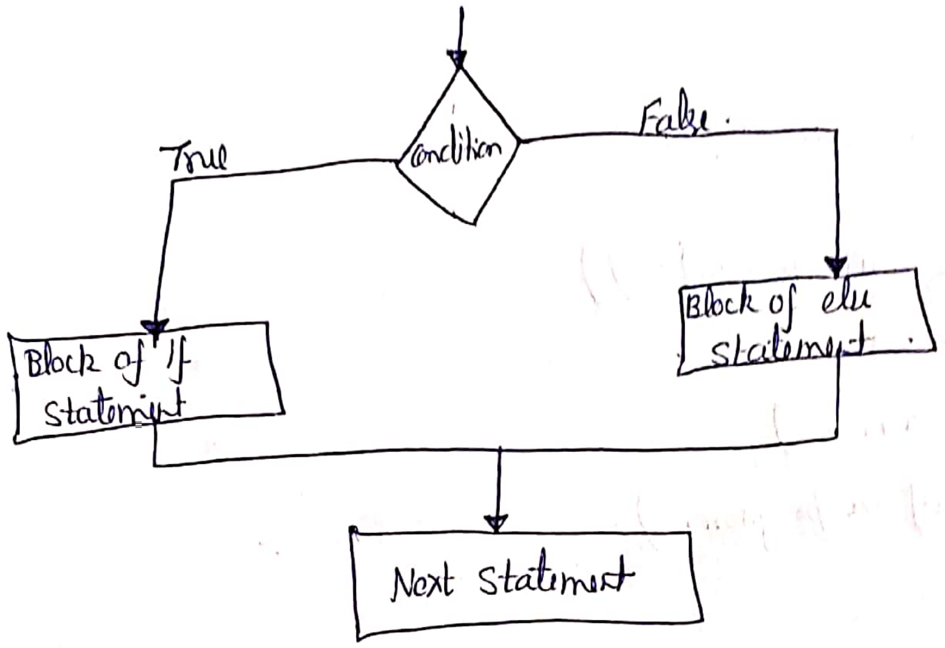
In case of if statement, the block of statements are executed only when the specified condition is true. But if the condition is false, then nothing is done and the control is transferred to the next statement following the if block. In case if some specific statements are to be executed in both the cases, then if_else condition is used.

Syntax

```
If condition:
    statement
else
    statement
```

// Program to find whether a number is even or odd.

```
n = int (input ("Enter the number"))
if n%2=0:
    print (n, "is even")
else :
    print (n, " is odd")
```



Fig! - Flowchart of if-else Statement

→ Nested Condition one conditional can also be nested within another.

for example! -

```
if x==y:
    print("x and y are equal")
else:
    if x<y:
        print("x is less than y")
    else:
        print("x is greater than y")
```

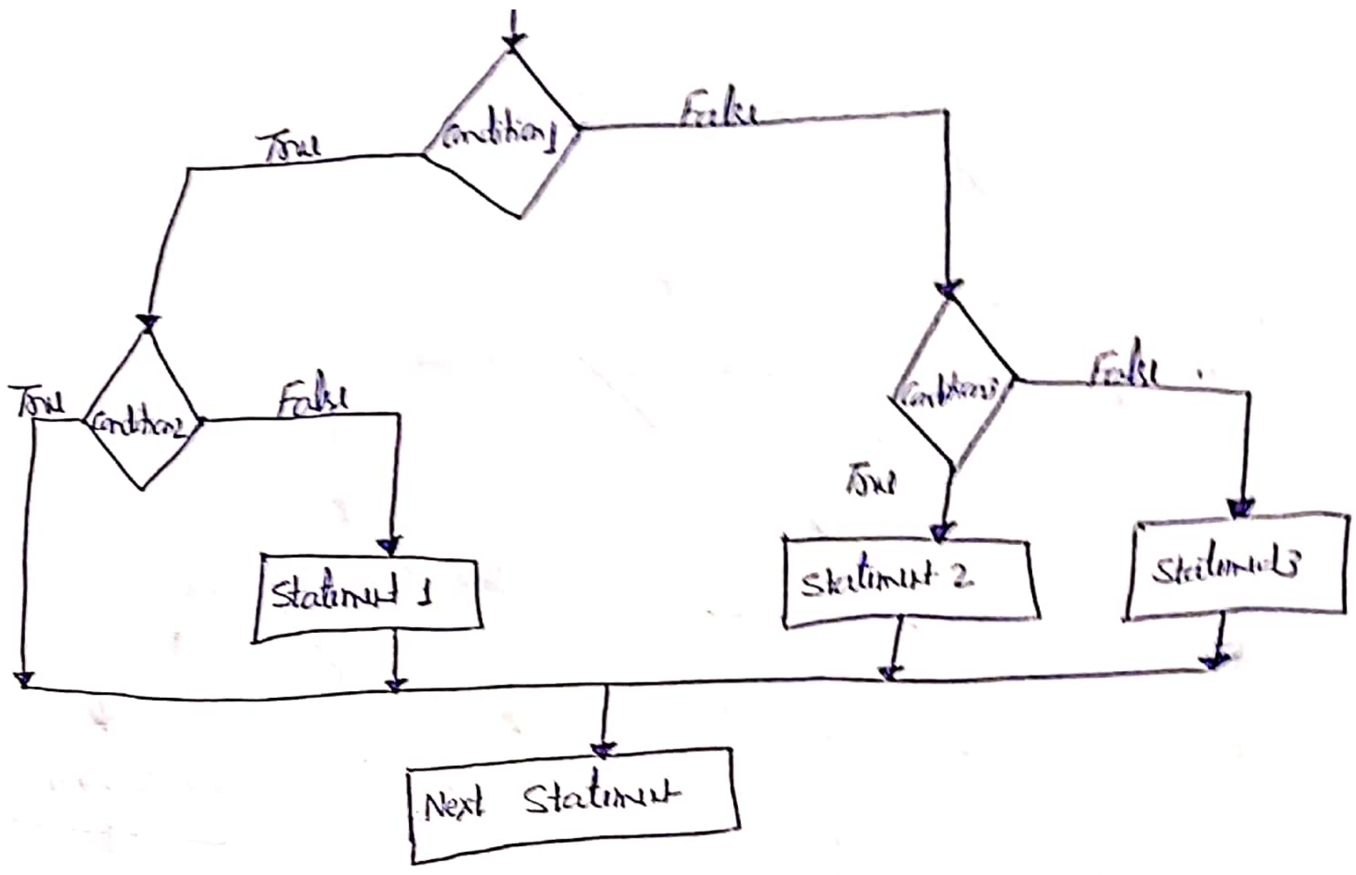


fig:- Flowchart of Nested if...else conditionals.

→ The if ....elif condition

The elif statement enables us to check multiple conditions and execute the specific block of statement depending upon the true condition among them.

Syntax

```
if expression 1:
    # block of statement.
elif expression 2:
    # block of Statement .
elif expression 3:
    # block of Statement
else :
    # block of Statement -
```

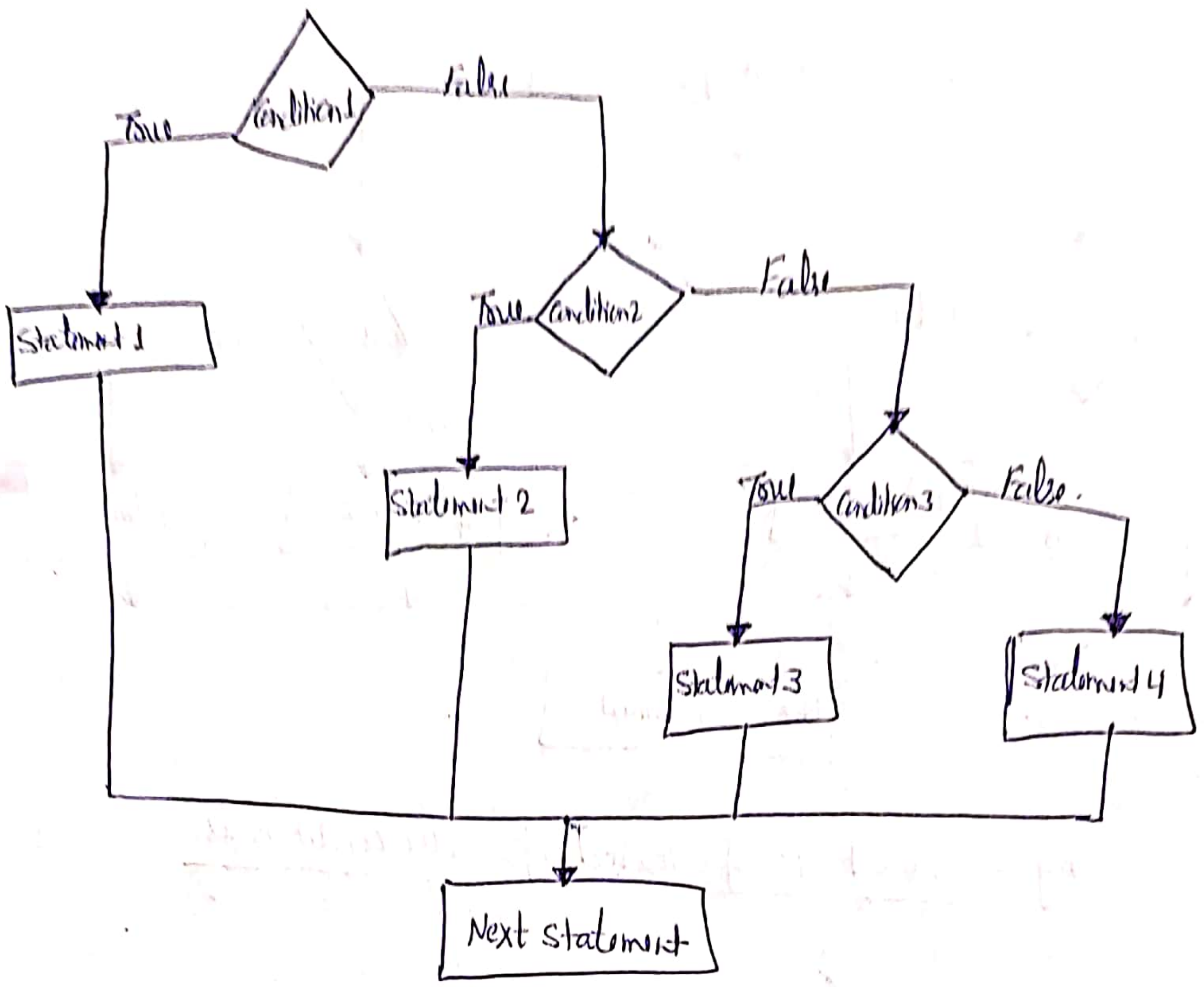


Fig:- The flowchart of the if...elif statement

Example:-

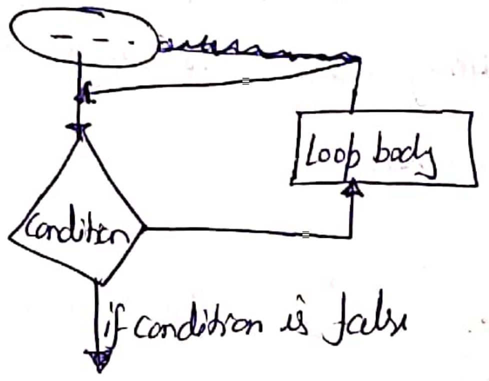
```
number = int(input ("Enter the number."))
if number = = 10:
        print("Number is equal to 10")

elif number = = 20:
        print(" Number is equal to 20")

elif number = = 30:
        print(" number is equal to 30")

else:
    print(" Number is not equal to (10,20,30)").
```

Python loops:- The flow of the program written in any programming language is sequential by default. Sometimes we may need to alter the flow of the program. The execution of a specific code may need to be repeated several number of times.



→ Why we use loops in Python?

The looping simplifies the complex problem into the easy one. It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times.

For example, if we need to print 10 natural number. then instead of using the print statement 10 times, we can print inside a loop which runs upto 10 iteration.

→ Advantages of Loops:-

- It provide code re-usability.
- Using loops, we do not need to write the same code again and again.

- Using loops, we can traverse over the elements of data structure (array or linked list).

→ FOR loop:

The for loop is used to iterate the statement as a part of the program several times. It is frequency used to traverse the data structure like List, tuple, Dictionary.

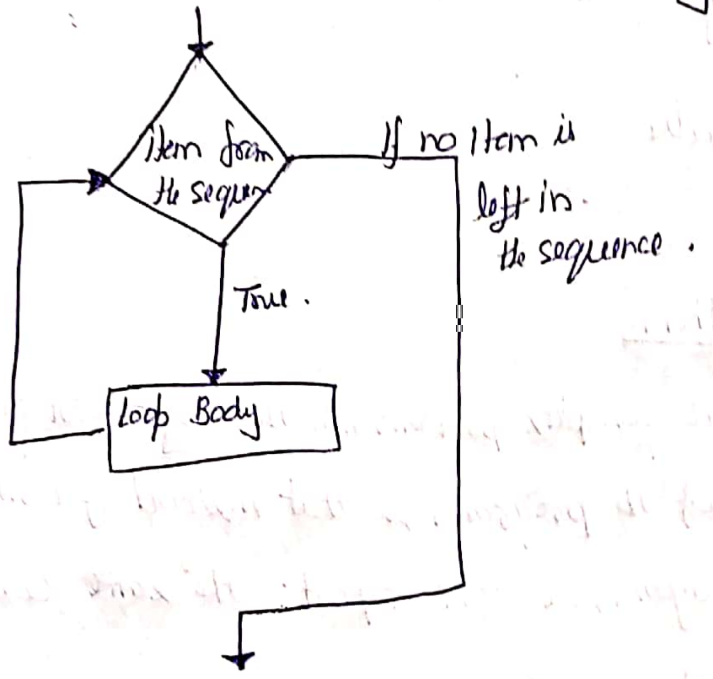


fig:- Flowchar of the for loop.

For example! - // program to print n even numbers

```
n=int(input("Enter the value of n"))
for i in range(0,n+1,2):   — step size.
    print(i)
```

Scanned by CamScanner

```
Enter the value of n 10·
0
2
4
6
8
10
>>>.
```

example! -2. // Program to print n to 1 in descending order.

```
n =int(input(" Enter the value of n: "))
for i in range(n, 0, -1)
    print(i)
```

```
Enter the value of n: 10
 10
 9
 8
 7
 6
 5
 4
 3
 2
 1
>>>
```

⟹ WHILE LOOP Statement :-

A while loop statement in Python Programming language repeatedly executes a target statement as long as a given condition is true.

Scanned by CamScanner

Syntax

```
while expression:
Statement(s)
```

The loop iterates while the condition is true.

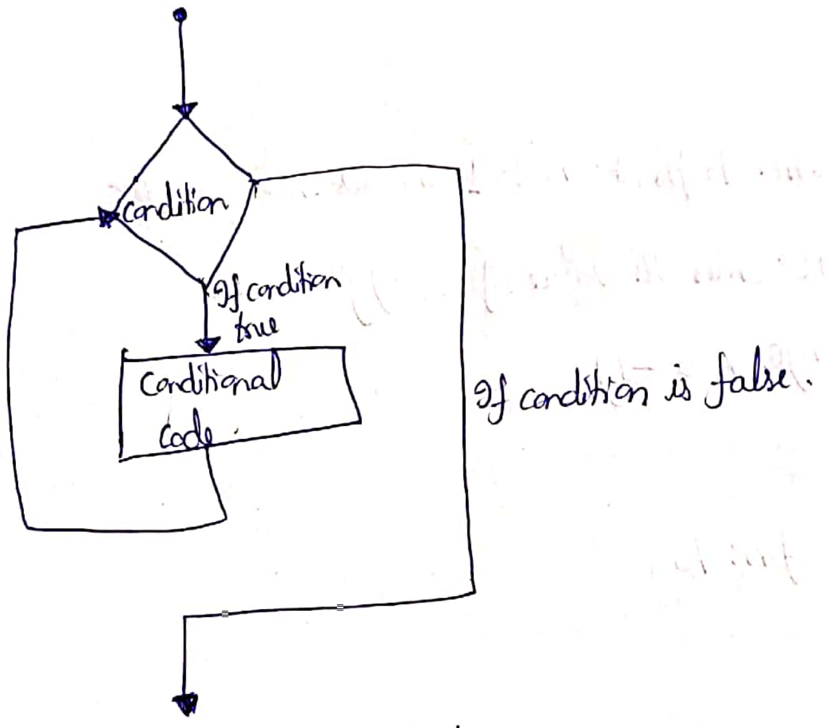


Fig:- Flow chart of while loop -

Example.

```
count = 0
while (count < 9) :
        print 'The count is : ', count
         Count = count + 1
print ("End of program").
```

o/p.

```
The count is : 1
 ,    ,   ,  : 2
 ,    ,   ,  : 3
 ,    ,   ,  : 4
 ,    ,   ,  : 5
 ,    ,   ,  : 6
 ,    ,   ,  : 7
 ,    ,   ,  : 8  → End of program.
```

→ The nested loop -

We can have a loop in the body of another loop. Loops can be nested in Python, as they can with other programming languages.

A nested loop is a loop that occurs within another loop.

* The syntax for a nested for loop is as follows:-

```
for iterating_var in sequence:
    for iterating_var in sequence:
        statement(s)
    statement(s).
```

* The syntax for a nested while loop statement is as following

```
While expression:
    While expression:
        statement(s)
    Statement(s)
```

Example - Program to Print the pattern using while loop.

```
n=int(input("Enter number of rows:"))
i=1
j=1
While i<=n
    While j<=i
        print("*", end="")
        j~~+~~=1
    j=1
    print()
    i+=1.
```

o/p ⇒
```
Enter number of rows: 4
*
* *
* * *
* * * *
>>>
```

Scanned by CamScanner

OR

```
// program to print Pattern using for loop
n = int(input("enter the number of rows: "))
for i in range(n)
    for j in range (i+1)
        print("*", end="")
    print() .
```

→ <u>The Break Statement</u>

The break statement is used to terminate the execution of a loop in which it is defined and the control moves to the next statement written immediately after the loop. The break statement is used within the while loop and for loop.

<u>Example:</u> -
```
// Program to illustrate the concept of break.
i=1
for i in range (20):
    if i==10:
        break
    print ("value of i is:", i)
print ("End of the Program")
```

```
o/p- value of i is: 1
     |   ||    |  | 2
                  3
     |   ||   ||  4
     |   ||   ||  5
                  6
                  7
                  8    ↗   "  "..: 9
                           End of program.
```

→ **The continue Statement**

In The continue statement, the current iteration is skipped and control shifts to the next iteration of the loop and skips the rest of the statements in the body of the loop.

Example:- // Program to illustrate the concept of continue.

```
i=1
for i in range(20):
   if i%2==0:
      continue
   print ("value of i is:", i)
print ("End of the program").
```

o/p→

```
Value of i is : 1
  |   |  | |  3
              5
            | 7
              9
  |   |  | |  11
              13
              15
  |   |  | |  17
              19
End of the program.
>>>
```