Unit II

CHAPTER 6

CONTROL STRUCTURES

BHMXUZ

Learning Objectives

After studying this lesson, students will be able to:

- To gain knowledge on the various flow of control in Python language.
- To learn through the syntax how to use conditional construct to improve the efficiency of the program flow.
- To apply iteration structures to develop code to repeat the program segment for specific number of times or till the condition is satisfied.

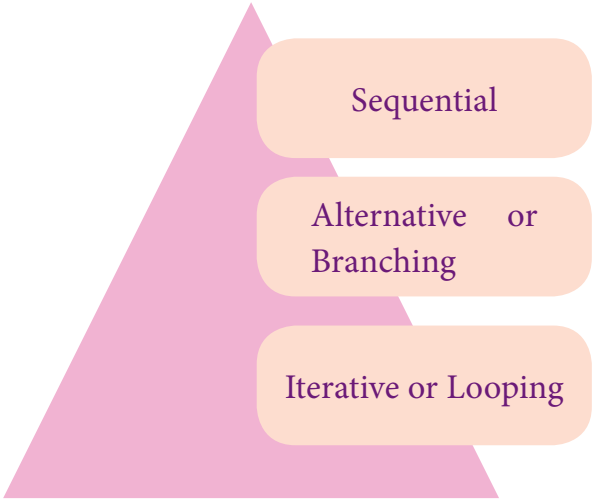
6.1 Introduction

Programs may contain set of statements. These statements are the executable segments that yield the result. In general, statements are executed sequentially, that is the statements are executed one after another. There may be situations in our real life programming where we need to skip a segment or set of statements and execute another segment based on the test of a condition. This is called ***alternative*** or ***branching***. Also, we may need to execute a set of statements multiple times, called ***iteration*** or ***looping***. In this chapter we are to focus on the various control structures in Python, their syntax and learn how to develop the programs using them.

6.2 Control Structures

A program statement that causes a jump of control from one part of the program to another is called **control structure** or **control statement.** As you have already learnt in C++, these control statements are compound statements used to alter the control flow of the process or program depending on the state of the process.

There are three important control structures



6.2.1 Sequential Statement

A **sequential statement** is composed of a sequence of statements which are executed one after another. A code to print your name, address and phone number is an example of sequential statement.

Example 6.1

```
# Program to print your name and address - example for sequential statement
print ("Hello! This is Shyam")
print ("43, Second Lane, North Car Street, TN")
```

Output

```
Hello! This is Shyam
43, Second Lane, North Car Street, TN
```

6.2.2 Alternative or Branching Statement

In our day-to-day life we need to take various decisions and choose an alternate path to achieve our goal. May be we would have taken an alternate route to reach our destination when we find the usual road by which we travel is blocked. This type of decision making is what we are to learn through alternative or branching statement. Checking whether the given number is positive or negative, even or odd can all be done using alternative or branching statement.

Python provides the following types of alternative or branching statements:

- **Simple if statement**
- **if..else statement**
- **if..elif statement**

(i) Simple if statement

Simple if is the simplest of all decision making statements. Condition should be in the form of relational or logical expression.

Syntax:

if <condition>:

statements-block1

In the above syntax if the condition is true statements - block 1 will be executed.

Example 6.2

```
# Program to check the age and print whether eligible for voting
x=int (input("Enter your age :"))
if x>=18:
        print ("You are eligible for voting")
```

Output 1:

```
Enter your age :34
You are eligible for voting
```

Output 2:

```
Enter your age :16
>>>
```

As you can see in the second execution no output will be printed, only the Python prompt will be displayed because the program does not check the alternative process when the condition is failed.

(ii) if..else statement

The **if .. else** statement provides control to check the true block as well as the false block. Following is the syntax of **'if..else'** statement.

Syntax:

if <condition>:

statements-block 1

else:

statements-block 2

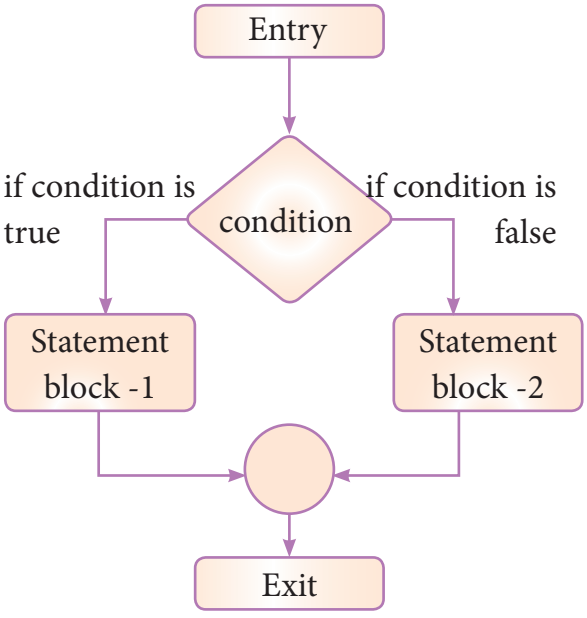


Fig. 6.1 if..else statement execution

if..else statement thus provides two possibilities and the condition determines which BLOCK is to be executed.

Example 6.3: #Program to check if the accepted number odd or even

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

Output 1:

```
Enter any number :56
56  is an even number
```

Output 2:

```
Enter any number :67
67  is an odd number
```

An alternate method to rewrite the above program is also available in Python. The complete **if..else** can also written as:

Syntax:

variable = variable1 if condition else variable 2

Note

The condition specified in the if is checked, if it is true, the value of variable1 is stored in variable on the left side of the assignment, otherwise variable2 is taken as the value.

Example 6.4: #Program to check if the accepted number is odd or even (using alternate method of if...else)

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

Output 1:

```
Enter any number :3
3 is odd
```

Output 2:

```
Enter any number :22
22 is even
```

(iii) Nested if..elif...else statement:

When we need to construct a chain of **if** statement(s) then **'elif'** clause can be used instead of **'else'.**

Syntax:

if <condition-1>:
    statements-block 1
elif <condition-2>:
    statements-block 2
else:
    statements-block n

In the syntax of **if..elif..else** mentioned above, condition-1 is tested if it is true then statements-block1 is executed, otherwise the control checks condition-2, if it is true statements-block2 is executed and even if it fails statements-block n mentioned in **else** part is executed.

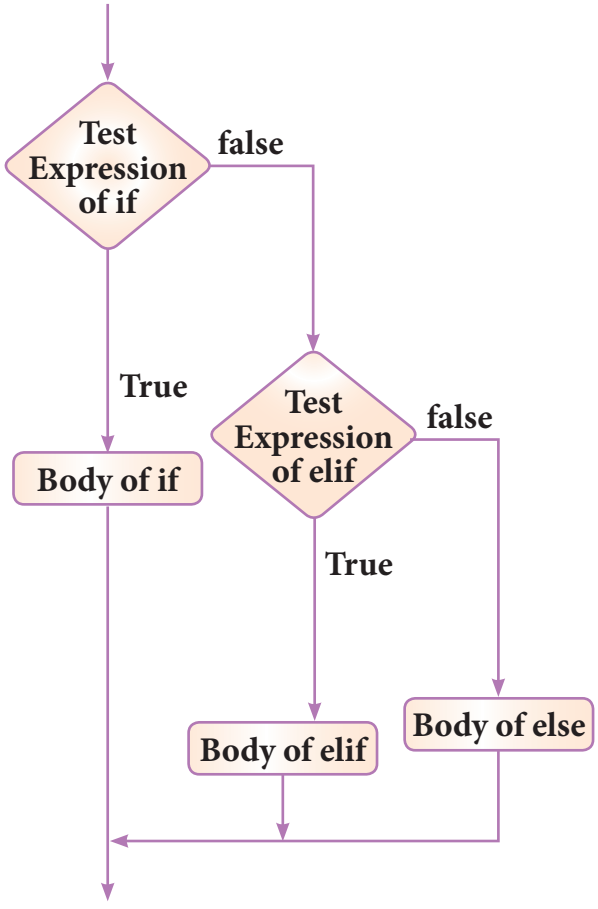


Fig 6.2 if..elif..else statement execution

'elif' clause combines **if..else-if..else** statements to one **if..elif…else. 'elif'** can be considered to be abbreviation of **'else if'.** In an **'if'** statement there is no limit of **'elif'** clause that can be used, but an **'else'** clause if used should be placed at the end.

Note

if..elif..else statement is similar to nested if statement which you have learnt in C++.

Example 6.5: #Program to illustrate the use of nested if statement

Average	***Grade***
>=80 and above	*A*
>=70 and <80	*B*
>=60 and <70	*C*
>=50 and <60	D
Otherwise	E

```
m1=int (input("Enter mark in first subject : "))
m2=int (input("Enter mark in second subject : "))
avg= (m1+m2)/2
if avg>=80:
        print ("Grade : A")
elif avg>=70 and avg<80:
        print ("Grade : B")
elif avg>=60 and avg<70:
        print ("Grade : C")
elif avg>=50 and avg<60:
        print ("Grade : D")
else:
        print ("Grade : E")
```

Output 1:

```
Enter mark in first subject : 34
Enter mark in second subject : 78
Grade : D
```

Output 2 :

```
Enter mark in first subject : 67
Enter mark in second subject : 73
Grade : B
```

Note

The two blocks of code in our example of if-statement are both **indented** four spaces, which is a typical amount of **indentation** for **Python**. In most other programming languages, **indentation** is used only to help make the code look pretty. But in **Python**, it is required to indicate to which block of code the statement belongs to.

Example 6.5a: #Program to illustrate the use of 'in' and 'not in' in if statement

```
ch=input ("Enter a character :")
# to check if the letter is vowel
if ch in ('a', 'A', 'e', 'E', 'i', 'I', 'o', 'O', 'u', 'U'):
        print (ch,' is a vowel')
# to check if the letter typed is not 'a' or 'b' or 'c'
if ch not in ('a', 'b', 'c'):
        print (ch,'  the letter is not a/b/c')
```

Output 1:

```
Enter a character :e
e is a vowel
```

Output 2:

```
Enter a character :x
x the letter is not a/b/c
```

6.2.3. Iteration or Looping constructs

Iteration or loop are used in situation when the user need to execute a block of code several of times or till the condition is satisfied. A **loop** statement allows to execute a statement or group of statements multiple times.

BX6SWB

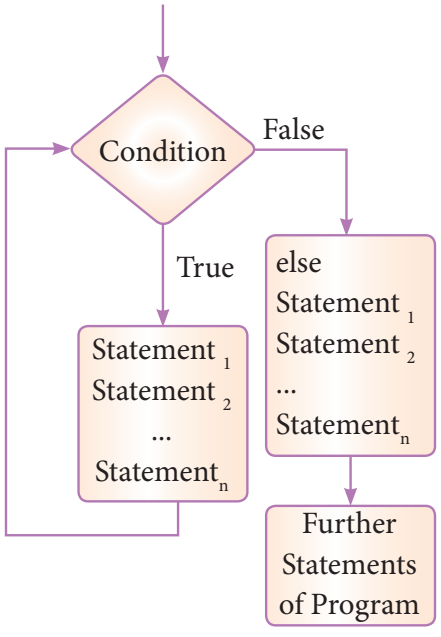


Fig 6.3 Diagram to illustrate how looping construct gets executed

Python provides two types of looping constructs:

- while loop
- for loop

(i) while loop

The syntax of while loop in Python has the following syntax:

Syntax:

while <condition>:
    statements block 1
[else:
    statements block2]

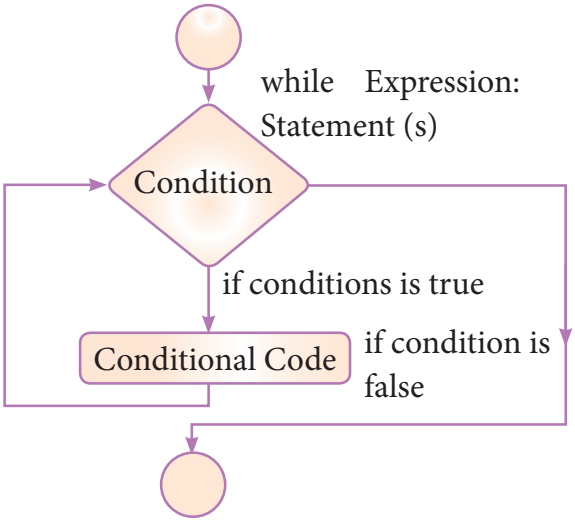


Fig 6.4 while loop execution

In the **while** loop, the condition is any valid Boolean expression returning True or False. The **else** part of while is optional part of **while**. The **statements block1** is kept executed till the condition is True. If the **else** part is written, it is executed when the condition is tested False. Recall **while** loop belongs to entry check loop type, that is it is not executed even once if the condition is tested False in the beginning.

Example 6.6: program to illustrate the use of while loop - to print all numbers from 10 to 15

```
i=10                    # intializing part of the control variable
while (i<=15):          # test condition
    print (i,end='\t')  # statements - block 1
    i=i+1               # Updation of the control variable
```

Output:

```
10    11    12    13    14    15
```

Note

That the control variable is **i**, which is initialized to 10, the condition is tested **i<=15,** if true value of **i** gets printed, then the control variable i gets updated as **i=i+1** (this can also be written as i +=1 using shorthand assignment operator). When **i** becomes **16**, the condition is tested False and this will terminate the loop.

Note

print can have end, sep as parameters. *end* parameter can be used when we need to give any escape sequences like '\t' for tab, '\n' for new line and so on. *sep* as parameter can be used to specify any special characters like, (comma) ; (semicolon) as separator between values (Recall the concept which you have learnt in previous chapter about the formatting options in print()).

Following is an example for using else part within while loop.

Example 6.7: program to illustrate the use of while loop - with else part

```
i=10                                          # intializing part of the control variable
while (i<=15):                                # test condition
      print (i,end='\t')                      # statements - block 1
      i=i+1                                   # Updation of the control variable
else:
    print ("\nValue of i when the loop exit ",i)
```

Output: 1

```
10   11   12   13   14   15
Value of i when the loop exit  16
```

(ii) for loop

for loop is the most comfortable loop. It is also an entry check loop. The condition is checked in the beginning and the body of the loop(statements-block 1) is executed if it is only True otherwise the loop is not executed.

Syntax:

```
for counter_variable in sequence:
      statements-block 1
[else:                          # optional block
      statements-block 2]
```

The *counter_variable* mentioned in the syntax is similar to the control variable that we

used in the **for** loop of C++ and the *sequence* refers to the initial, final and increment value. Usually in Python, **for** loop uses the *range()* function in the sequence to specify the initial, final and increment values. *range()* generates a list of values starting from **start** till **stop-1.**

The syntax of range() is as follows:

range (start,stop,[step])

Where,

start – refers to the initial value

stop – refers to the final value

step – refers to increment value, this is optional part.

Example 6.8: Examples for range()

range (1,30,1)	*will start the range of values from 1 and end at 29*
range (2,30,2)	*will start the range of values from 2 and end at 28*
range (30,3,-3)	*- will start the range of values from 30 and end at 6*
range (20)	*will consider this value 20 as the end value(or upper limit) and starts the range count from 0 to 19 (remember always range() will work till stop -1 value only)*

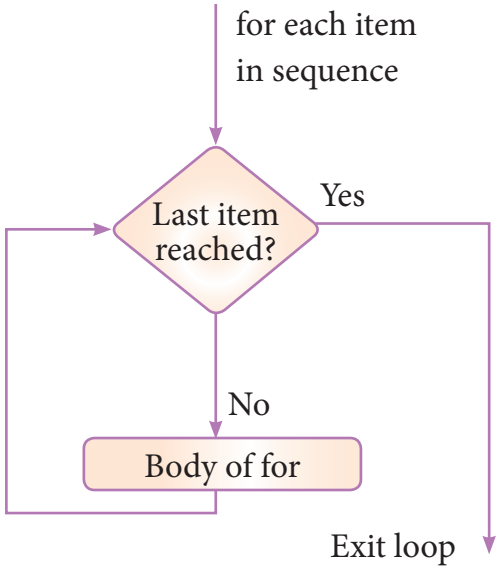


Fig 6.5 for loop execution

Example 6.9: #program to illustrate the use of for loop - to print single digit even number

```
for i in range (2,10,2):
        print (i, end=' ')
```

Output:

2 4 6 8

Following is an illustration using else part in for loop

Example 6.10 : #program to illustrate the use of for loop - to print single digit even number with else part

```
for i in range(2,10,2):
        print (i,end=' ')
else:
        print ("\nEnd of the loop")
```

Output:

```
2 4 6 8
End of the loop
```

Note

In Python, indentation is important in loop and other control statements. Indentation only creates blocks and sub-blocks like how we create blocks within a set of { } in languages like C, C++ etc.

Here is another program which illustrates the use of range() to find the sum of numbers 1 to 100

Example 6.11: # program to calculate the sum of numbers 1 to 100

```
n = 100
sum = 0
for counter in range(1,n+1):
        sum = sum + counter
print("Sum of 1 until %d: %d" % (n,sum))
```

Output:

Sum of 1 until 100: 5050

In the above code, *n* is initialized to 100, *sum* is initialized to 0, the *for* loop starts executing from 1, for every iteration the value of sum is added with the value of counter variable and stored in sum. Note that the for loop will iterate from 1 till the upper limit -1 (ie. Value of n is set as 100, so this loop will iterate for values from 1 to 99 only, that is the reason why we have set the upper limit as n+1)

Note

range () can also take values from string, list, dictionary etc. which will be dealt in the later chapters.

Following is an example to illustrate the use of string in range()

Example 6.12: program to illustrate the use of string in range() of for loop

```
for word in 'Computer':
    print (word,end=' ')
else:
    print ("\nEnd of the loop")
```

Output

```
C o m p u t e r
End of the loop
```

(iii) Nested loop structure

A loop placed within another loop is called as nested loop structure. One can place a **while** within another **while; for** within another **for; for** within **while** and **while** within **for** to construct such nested loops.

Following is an example to illustrate the use of for loop to print the following pattern

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

Example 6.13: program to illustrate the use nested loop -for within while loop

```
i=1
while (i<=6):
    for j in range (1,i):
        print (j,end='\t')
    print (end='\n')
    i +=1
```

Output:

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

6.2.4 Jump Statements in Python

The jump statement in Python, is used to unconditionally transfer the control from one part of the program to another. There are three keywords to achieve jump statements in Python : **break, continue, pass.** The following flowchart illustrates the use of break and continue.

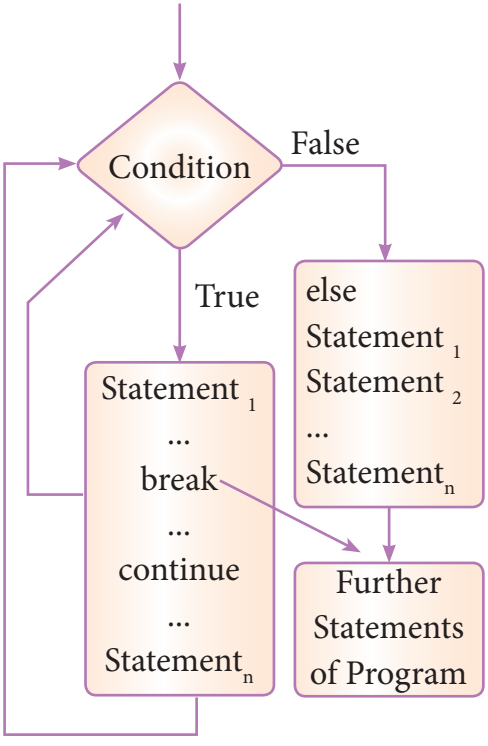


Fig 6.6 Use of break, continue statement in loop structure

(i) break statement

The **break** statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.

A **while** or **for** loop will iterate till the condition is tested false, but one can even transfer the control out of the loop (terminate) with help of **break** statement. When the break statement is executed, the control flow of the program comes out of the loop and starts executing the segment of code after the loop structure.

If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.

Syntax:

break

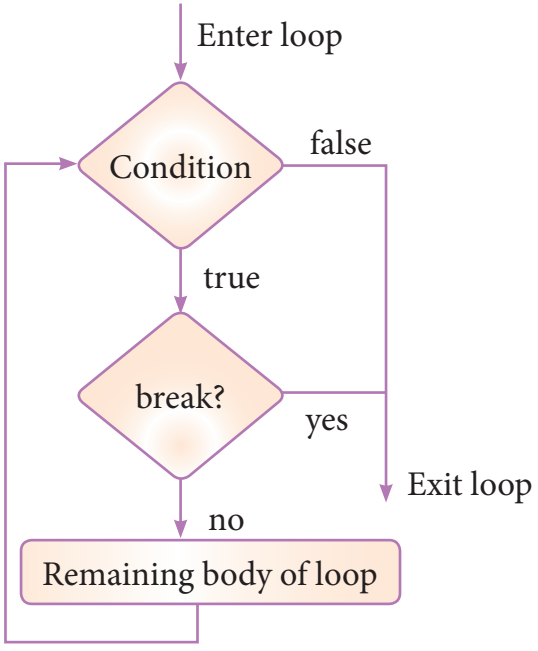


Fig 6.7 Working of break statement

The working of break statement in **for** loop and **while** loop is shown below.

```
for var in sequence:
        if condition:
                break
                #code inside for loop
#code outside for loop
while test expression:
        #code inside while loop
        if condition:
            break
        #code inside while loop
#code outside while loop
```

Example 6.14: Program to illustrate the use of break statement inside for loop

```
for word in "Jump Statement":
    if word = = "e":
        break
        print (word, end= ' ')
```

Output:

```
Jump Stat
```

The above program will repeat the iteration with the given "Jump Statement" as string. Each letter of the given string sequence is tested till the letter 'e' is encountered, when it is encountered the control is transferred outside the loop block or it terminates. As shown in the output, it is displayed till the letter 'e' is checked after which the loop gets terminated.

One has to note an important point here is that **'if a loop is left by break, the else part is not executed'.** To explain this lets us enhance the previous program with an 'else' part and see what output will be:

Example 6.15: Program to illustrate the use of break statement inside for loop

```
for word in "Jump Statement":
    if word = = "e":
        break
        print (word, end="")
else:
        print ("End of the loop")
print ("\n End of the program")
```

Output:

```
Jump Stat
End of the program
```

Note that the **break** statement has even skipped the 'else' part of the loop and has transferred the control to the next line following the loop block.

(ii) continue statement

Continue statement unlike the break statement is used to skip the remaining part of a loop and start with next iteration.

Syntax:

continue

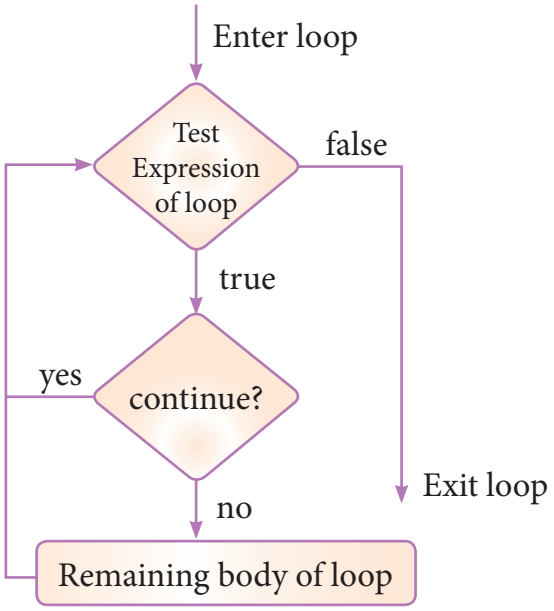


Fig 6.8 Working of continue statement

The working of continue statement in **for** and **while** loop is shown below.

```
for var in sequence:
        # code inside for loop
        if condition:
                continue
                #code inside for loop
#code outside for loop
while test expression:
        #code inside while loop
        if condition:
                continue
                #code inside while loop
#code outside while loop
```

Example 6.16: Program to illustrate the use of continue statement inside for loop

```
for word in "Jump Statement":
    if word = = "e":
        continue
    print (word, end = ' ')
print ("\n End of the program")
```

Output:

```
Jump Statmnt
End of the program
```

The above program is same as the program we had written for 'break' statement except that we have replaced it with 'continue'. As you can see in the output except the letter 'e' all the other letters get printed.

(iii) pass statement

pass statement in Python programming is a null statement. pass statement when executed by the interpreter it is completely ignored. Nothing happens when pass is executed, it results in no operation.

pass statement can be used in **'if'** clause as well as within loop construct, when you do not want any statements or commands within that block to be executed.

Syntax:

pass

Example 6.17: Program to illustrate the use of pass statement

```
a=int (input("Enter any number :"))
if (a==0):
    pass
else:
    print ("non zero value is accepted")
```

Output:

```
Enter any number :3
non zero value is accepted
```

When the above code is executed if the input value is 0 (zero) then no action will be performed, for all the other input values the output will be as follows:

Note

pass statement is generally used as a placeholder. When we have a loop or function that is to be implemented in the future and not now, we cannot develop such functions or loops with empty body segment because the interpreter would raise an error. So, to avoid this we can use pass statement to construct a body that does nothing.

Example 6.18: Program to illustrate the use of pass statement in for loop

```
for val in "Computer":
        pass
print ("End of the loop, loop structure will be built in future")
```

Output:

End of the loop, loop structure will be built in future.

Points to remember:

- Programs consists of statements which are executed in sequence, to alter the flow we use control statements.
- A program statement that causes a jump of control from one part of the program to another is called control structure or control statement.
- Three types of flow of control are
  - o Sequencing
  - o Branching or Alternative
  - o Iteration
- In Python, branching is done using various forms of 'if' structures.
- Indentation plays a vital role in Python programming, it is the indentation that group statements no need to use {}.
- Python Interpreter will throw error for all indentation errors.
- To accept input at runtime, earlier versions of Python supported raw_input(), latest versions support input().
- print() supports the use of escape sequence to format the output to the user's choice.
- range() is used to supply a range of values in for loop.
- break, continue, pass act as jump statements in Python.
- pass statement is a null statement, it is generally used as a place holder.

Hands on Experience

1. Write a program to check whether the given character is a vowel or not.
2. Using if..else..elif statement check smallest of three numbers.
3. Write a program to check if a number is Positive, Negative or zero.
4. Write a program to display Fibonacci series 0 1 1 2 3 4 5…… (upto n terms)
5. Write a program to display sum of natural numbers, upto n.
6. Write a program to check if the given number is a palindrome or not.
7. Write a program to print the following pattern

```
* * * * *
* * * *
* * *
* *
*
```

8. Write a program to check if the year is leap year or not.

Evaluation

Part - I

Choose the best answer **1 Marks**

1. How many important control structures are there in Python?
   A) 3 B) 4
   C) 5 D) 6
2. elif can be considered to be abbreviation of
   A) nested if B) if..else
   C) else if D) if..elif
3. What plays a vital role in Python programming?
   A) Statements B) Control
   C) Structure D) Indentation
4. Which statement is generally used as a placeholder?
   A) continue B) break
   C) pass D) goto

CCPMXM

5. The condition in the if statement should be in the form of

   A) Arithmetic or Relational expression

   B) Arithmetic or Logical expression

   C) Relational or Logical expression

   D) Arithmetic

6. Which is the most comfortable loop?

   A) do..while B) while

   C) for D) if..elif

7. What is the output of the following snippet?

```
i=1
while True:
        if i%3 ==0:
                break
        print(i,end='')
        i +=1
```

   A) 12 B) 123

   C) 1234 D) 124

8. What is the output of the following snippet?

```
T=1
while T:
        print(True)
        break
```

   A) False B) True

   C) 0 D) 1

9. Which amongst this is not a jump statement ?

   A) for B) pass

   C) continue D) break

10. Which punctuation should be used in the blank?

```
if <condition>_
   statements-block 1
else:
   statements-block 2
```

   A) ; B) :

   C) :: D) !

Part -II

Answer the following questions **2 Marks**

1. List the control structures in Python.
2. Write note on break statement.
3. Write is the syntax of if..else statement
4. Define control structure.
5. Write note on range () in loop

Part -III

Answer the following questions **3 Marks**

1. Write a program to display

   A

   A B

   A B C

   A B C D

   A B C D E
2. Write note on if..else structure.
3. Using if..else..elif statement write a suitable program to display largest of 3 numbers.
4. Write the syntax of while loop.
5. List the differences between break and continue statements.

Part -IV

Answer the following questions **5 Marks**

1. Write a detail note on for loop
2. Write a detail note on if..else..elif statement with suitable example.
3. Write a program to display all 3 digit odd numbers.
4. Write a program to display multiplication table for a given number.