Unit 3
Control Flow, Functions

Conditionals: Boolean values and operators, conditional (if), alternative (if-else), chained conditional (if-elif-else); Iteration: state, while, for, break, continue, pass; Fruitful functions: return values, parameters, local and global scope, function composition, recursion; Strings: string slices, immutability, string functions and methods, string module; Lists as arrays. Illustrative programs: square root, gcd, exponentiation, sum an array of numbers, linear search, binary search.

3.1 BOOLEAN VALUES AND OPERATORS

The Boolean data type is a data type, having two values (usually denoted true and false), When converting a bool to an int, the integer value is always 0 or 1, but when converting an int to a bool, the boolean value is True for all integers except 0.

3.1.1 Boolean and Logical Operators

Boolean values respond to logical operators and / or
True and False = False
True and True = True
False and True = False
False or True = True
False or False = False

3.2 CONDITIONAL STATEMENTS IN PYTHON

In programming and scripting languages, conditional statements are used to perform different computations or actions depending on whether a condition evaluates to true or false. (In python, it is written as True and False). Usually, the condition uses comparisons and arithmetic expressions with variables. These expressions are evaluated to the Boolean values True or False. The statements for the decision taking are called conditional statements. They are also known as conditional expressions or conditional constructs. Following is the general form of a typical conditional checking or decision making structure found in most of the programming languages.

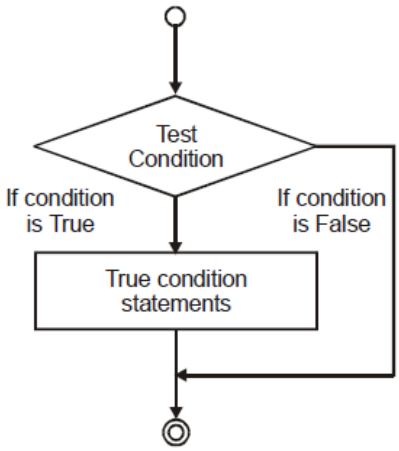

Figure 3.1 Decision making structure

Types of Conditional Checking Statements in Python

Python programming language provides following types of conditional checking statements.

1. if statements
2. if...else statements
3. if...elif...else statements(or) chained conditionals
4. Nested if statements

Rules

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

3.2.1 if Statements

The program evaluates the test condition and executes the statement(s) only if the text condition is True. If the text condition is False, the statement(s) are not executed.

Syntax:

```
if condition :
        True Statement Block
```

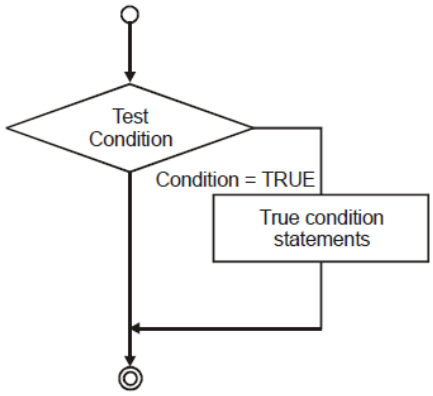

Figure 3.2 Flowchart for if statement

PROGRAM 1

Example program for if statement

```
1 age = input("What's your age = ")
2
3 if ( age >= 60 ) :
4 print "You are a Senior Citizen"
5 print "Good bye!"
```

EXECUTION

```
sh-4.3$ python program1.py
What's your age = 68
You are a Senior Citizen
Good bye!
```

3.2.2 if...else Statements

The if..else statement evaluates test condition and executes the true statement block only when test condition is True. If the condition is False, false statement block is executed. Indentation is used to separate the blocks.

Syntax:

```
if condition:
        True Statement Block
else:
        False Statement Block
```

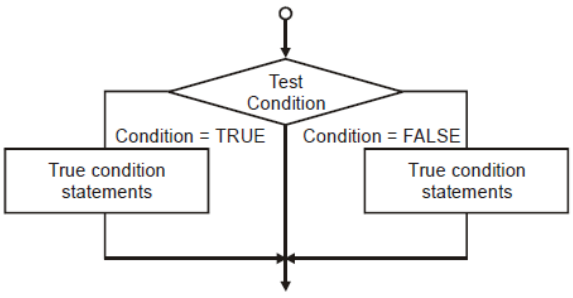

Figure 3.3 Flowchart for if...else Statements

PROGRAM 2

Example program for if...else statement

```
1 age = input("What's your age = ")
2
3 if ( age >= 60 ) :
4 print "Sorry! You are a Senior Citizen"
5 print "Good bye!"
6 else:
7 print "You are selected"
8 print "Have a great day!"
```

EXECUTION 1

```
sh-4.3$ python program2.py
What's your age = 45
You are selected
Have a great day!
```

EXECUTION 2

```
sh-4.3$ python program2.py
What's your age = 63
Sorry! You are a Senior Citizen
Good bye!
```

PROGRAM 3

```
1 # This program compares two strings.
2
3 password =raw_input('Enter the password: ')
4
5 if password == 'python':
6 print'Password Accepted'
7 else:
8 print'Sorry, you have entered wrong password.'
```

EXECUTION 1

```
sh-4.3$ python program3.py
Enter the password: python
Password Accepted
```

EXECUTION 2

```
Enter the password: ji
Sorry, you have entered wrong password.
```

Ternary Operator: C and Python

The ternary operator(? ; ;) in C takes three arguments. The first argument is a comparison argument, the second is the result upon a true comparison, and the third is the result upon a false comparison.

Syntax:

if_condition ? value_if_true : value_if_false

Example:

max = (a > b) ? a : b;

The above code is equivalent to following code:

```
if (a > b):
    max=a
else:
    max=b
```

In Python, the syntax for ternary operator is

Syntax 1:

value_if_true_(if_condition) else value_if_false

Syntax 2:

(value_if_true, value_if_false) [if_condition]

Example:

max = a if (a > b) else b (or) max = (a,b)[(a > b)]

PROGRAM 4

```
1 a = raw_input("Enter a = ")
2 b = raw_input("Enter b = ")
3
4 max = (a,b)[(a > b)]
5 max = a if (a > b) else b # work in some compilers only
6 print "Maximum of two numbers = ", max
```

EXECUTION 1

```
sh-4.3$ python program4.py
Enter a = 89
Enter b = 568
Maximum of two numbers = 568
```

EXECUTION 2

```
Enter a = 56
Enter b = 41
Maximum of two numbers = 41
```

3.2.3 if...elif...else (or) Chained Conditionals

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

Syntax:

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

Flowchart for if...elif...else Statements

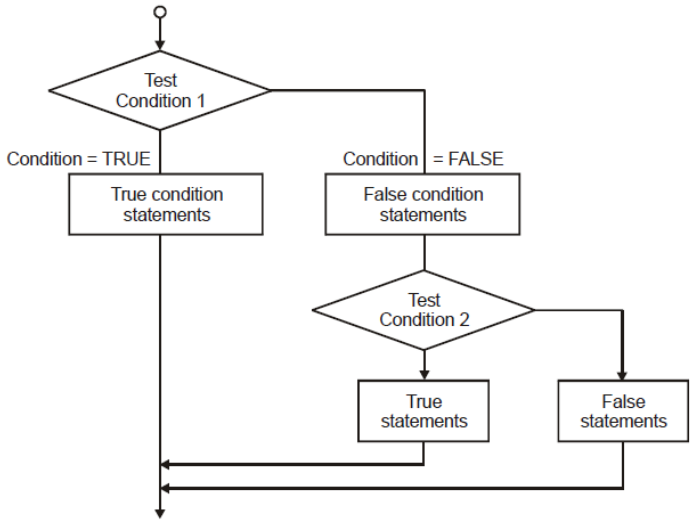

Figure 3.4 Flowchart forif...elif...else Statements

PROGRAM 5

Example program for if...elif...else Statement

```
1 age = input("What's your age = ")
2
3 if (age >= 60):
4 print "Sorry! You are a Senior Citizen"
5 print "Good bye!"
6 elif (age < 60 and age > 40):
7 print "You are selected for the post of Senior Manager"
8 print "Have a great day!"
9 elif (age < 40 and age > 20):
10 print "You are selected for the post of Junior Manager"
11 print "Have a great day!"
12 else:
13 print("Sorry! you cannot apply")
```

EXECUTION 1

sh-4.3$ python program5.py
What's your age = 72
Sorry! You are a Senior Citizen
Good bye!
EXECUTION 2
What's your age = 55
You are selected for the post of Senior Manager
Have a great day!
EXECUTION 3
What's your age = 18
Sorry! you cannot apply

3.2.4 Nested if Statements

if...elif...else statement can be used inside another if...elif...else statement. This is called nesting in computer programming.

PROGRAM 6

Example program for Nested if Statements

```
1 #check whether the given number is positive, negative or zero
2 number = input("Enter a number: ")
3 if number >= 0:
4 if number == 0:
5 print("Zero")
6 else:
7 print("Positive number")
8 else:
9 print("Negative number")
```

EXECUTION 1
sh-4.3$ python program6.py
Enter a number: 8
Positive number
EXECUTION 2
Enter a number: -9
Negative number
EXECUTION 3
Enter a number: 0
Zero

PROGRAM 7

```
1 mark1 = input("Enter Mark 1 = ")
2 mark2 = input("Enter Mark 2 = ")
3 mark3 = input("Enter Mark 3 = ")
4 total_marks = mark1 + mark2 + mark3
```

```
5 print "Total Marks obtained =", total_marks
6 Average = total_marks/3
7 print "Average =", Average
8 if Average >= 90:
9 grade = 'A'
10 else:
11 if Average >= 80:
12 grade = 'B'
13 else:
14 if Average >= 70:
15 grade = 'C'
16 else:
17 if Average >= 60:
18 grade = 'D'
19 else:
20 grade = 'F'
21 print "Grade of the Student = ", grade
```

EXECUTION 1

```
sh-4.3$ python program7.py
Enter Mark 1 = 91
Enter Mark 2 = 96
Enter Mark 3 = 89
Total Marks obtained = 276
Average = 92
Grade of the Student = A
```

EXECUTION 2

```
Enter Mark 1 = 23
Enter Mark 2 = 45
Enter Mark 3 = 31
Total Marks obtained = 99
Average = 33
Grade of the Student = F
```

3.3 ITERATION (OR) LOOPING

Computers often do repetitive tasks. There are situations when programmers need to execute a block of code several number of times. Repeated execution of a set of statements is called iteration or looping.

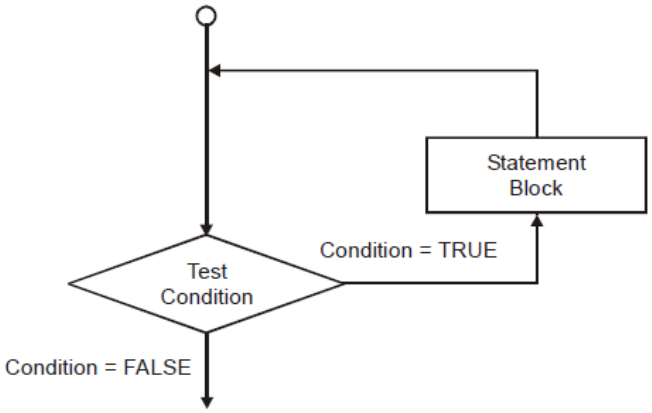

Figure 3.5 Flowchart for looping statements

Python programming language provides following types of iterative statements.

1. The for loop
2. The while statement
3. Nested loops

3.3.1 The for Loop

For loop in Python repeats a group of statements for a specified number of times. It is used with Python's sequence data types - strings, lists, and tuples. For loop in python starts with the keyword "for" followed by an arbitrary variable name, which holds its values in the following sequence object.

Syntax:

for <loop_variable> in <sequence>:
    <statement or statement block)>
else
    <statements or statement block>

The else block is special in python and it's an unknown concept to C and C++ programming. The else block will be executed only if the for loop hasn't been broken by a "break" statement.

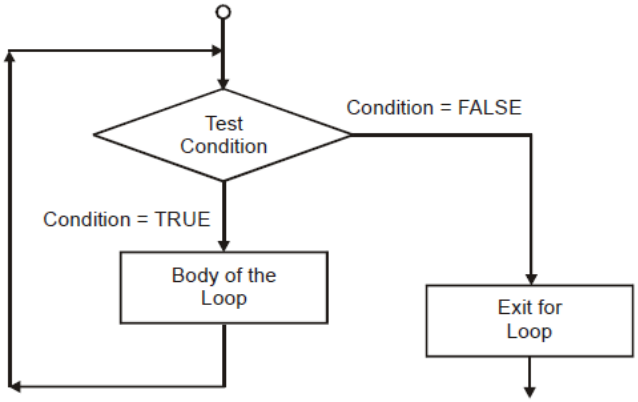

Figure 3.6 Flowchart - for Loop

PROGRAM 8

Example program for FOR loop:

```
1 # simple program to explain for loop
2 for i in '123':
3 print "Welcome",i,"times"
```

EXECUTION

```
sh-4.3$ python program8.py
Welcome 1 times
Welcome 2 times
Welcome 3 times
```

PROGRAM 9

```
1 # Program to find the sum of all numbers stored in a list
2 numbers = [5, 3, 8, 4, 26, 5, 4, 11, 3, 89]
3 sum = 0
4
5 # iterate over the list
6 for val in numbers:
7 sum = sum+val
8 print "Sum of all numbers stored in the list = ", sum
```

EXECUTION

```
sh-4.3$ python program9.py
Sum of all numbers stored in the list = 158
```

PROGRAM 10

```
1 # Program to find the sum of all numbers stored in a list
2 numbers = [5, 3, 8, 4, 26, 5, 4, 11, 3, 89]
3 sum = 0
4
5 # iterate over the list
6 for val in numbers:
7 sum = sum+val
8 print "Sum of all numbers stored in the list = ", sum
```

EXECUTION

```
sh-4.3$ python program10.py
Sum of all numbers stored in the list = 158
```

1. The Range() Function

Sequence of numbers are also generated using range() function. range() is defined with start_element, stop_element and step size. Default step size is equal to 1 if not provided. The range() function is useful in combination with the for loop.

Syntax:

range(start_element, stop_element, step size)

PROGRAM 11

```
1 # Display numbers from 0 to 10 (11 numbers)
2 numbers = range(11)
3 print(numbers)
4
5 # Display numbers from 5 (start_element) to 10
(stop_element)
6 numbers = range(5,11)
7 print(numbers)
8
9 # Display numbers from 1 (start_element) to
10 # 11 (stop_element) with step_size = 2
11 numbers = range(1,11,2)
12 print(numbers)
13
14 # Display numbers from 10 (start_element) to
15 # 0 (stop_element) with step_size = -1
16 numbers = range(10, 0, -1)
17 print(numbers)
```

EXECUTION

```
sh-4.3$ python program11.py
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[5, 6, 7, 8, 9, 10]
[1, 3, 5, 7, 9]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

PROGRAM 12

```
1 n = 11
2 sum = 0
3 for i in range(1,n):
4 sum = sum + i
5 print "Sum of ",n," numbers =", sum
```

EXECUTION

```
sh-4.3$ python program12.py
Sum of 11 numbers = 55
```

PROGRAM 13

```
1 items = ['pen', 'paper', 'pencil', 'eraser']
2 count = len(items) #count the number of items in
the list
```

```
3 print "Number of items in the list = ", count
4
5 for i in range(count):
6 print "Item ", i+1 ," = ", items[i]
```

EXECUTION

```
sh-4.3$ python program13.py
Number of items in the list = 4
Item 1 = pen
Item 2 = paper
Item 3 = pencil
Item 4 = eraser
```

PROGRAM 14

```
1 # for loop with else
2
3 computer_brands = ["Apple", "Asus", "Dell", "Samsung"]
4 for i in computer_brands:
5 print i
6 else:
7 print("No new brands")
```

EXECUTION

```
sh-4.3$ python program14.py
Apple
Asus
Dell
Samsung
No new brands
```

3.3.2 The while Loop

The while loop in Python iterates over a block of statements as long as the test condition is true. The statement body is entered only when the test condition is true. After first iteration, the test condition is checked again. The process continues until the test condition evaluates to False. In Python, the statement body is determined through indentation. Python interprets any non-zero value as True. None and 0 are interpreted as False. Python supports to have an else statement with a while loop statement. The else statement is executed when the condition becomes false.

Difference between for Loop and while Loop

Both for Loop and while Loops are used for repeating sections of code. But unlike a For loop, the while loop will not run n times. It runs until a defined test condition is met. If the test condition is false, it comes out of the loop.

Syntax:

```
while condition:
    statement_1
    ...
    statement_n
```

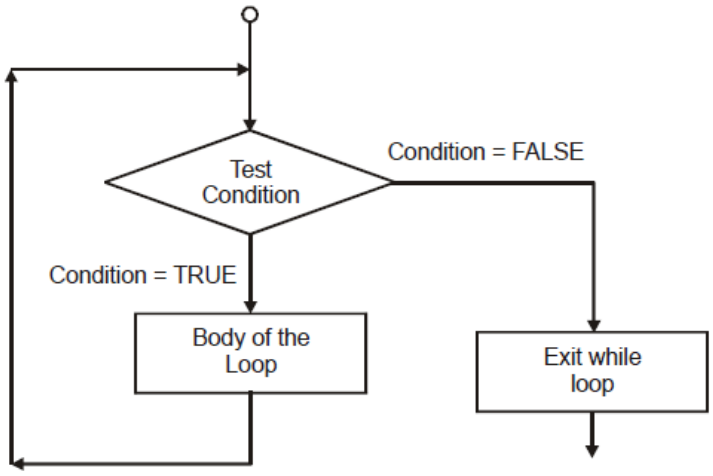

Figure 3.7 Flowchart - while Loop

PROGRAM 15

Example program for While loop

```
1 # Simple While loop
2
3 a = 0
4 #Body of the loop is intended by four spaces
5 while a < 10:
6 a = a + 1
7 print a
```

EXECUTION

```
sh-4.3$ python program15.py
1
2
3
4
5
6
7
8
9
10
```

PROGRAM 16

```
1 # Simple While loop - last print statement is not intended
2
3 a = 0
4 while a < 10:
5 a = a + 1
6 print a
```

EXECUTION

```
sh-4.3$ python program16.py
10
```

Syntax:

```
While - Else loop
        while condition:
                statement_1
                ....
                statement_n
        else
                statement_2
                ...
                statement_n
```

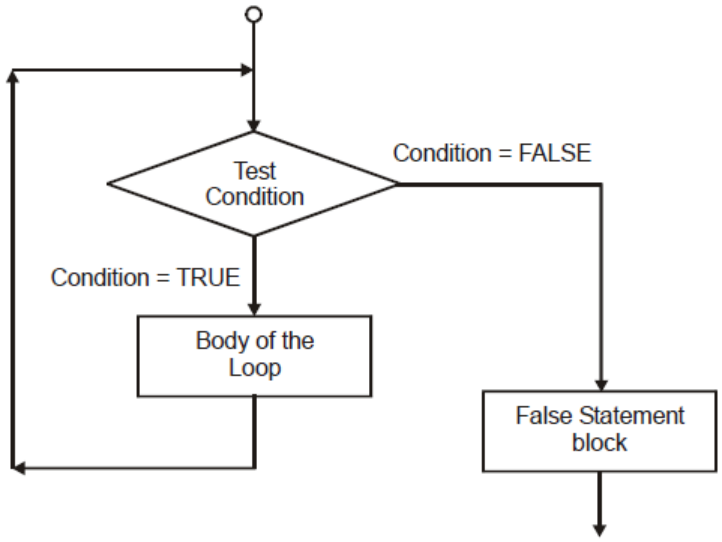

Figure 3.8 Flowchart for while-else Loop

PROGRAM 17

Example program for While-Else loop

```
1 #program for While-Else statement
2 count = 0
3 while count < 5:
```

```
4 print count, " is less than 5"
5 count = count + 1
6 else:
7 print count, " is not less than 5"
8
```

EXECUTION

```
sh-4.3$ python program17.py
0 is less than 5
1 is less than 5
2 is less than 5
3 is less than 5
4 is less than 5
5 is equal to 5
```

PROGRAM 18

```
1 computer_brands = ["Apple", "Asus", "Dell", "Samsung"]
2 i = 0
3
4 while i <len(computer_brands):
5 print computer_brands[i]
6 i = i + 1
```

EXECUTION

```
sh-4.3$ python program18.py
Apple
Asus
Dell
Samsung
```

3.3.3 Nested Loops Statements

Python programming language allows using one loop inside another loop.

Syntax:

Nested for loop

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

Syntax:

Nested while loop

```
while expression:
    while expression:
        statement(s)
statement(s)
```

PROGRAM 19

Example program for Nested loops

```
1 #program to explain for loop
2 list1 = [["john", "Jerry"],[1, 2], ["Paris", "London"]]
3
4 for i in list1:
5 print i
```

EXECUTION

```
sh-4.3$ python program19.py
['john', 'Jerry']
[1, 2]
['Paris', 'London']
```

PROGRAM 20

```
1 #program to show nested loops -for loop within
another for loop
2 list1 = [["john", "Jerry"],[1, 2], ["Paris", "London"]]
3
4 for i in list1:
5 for x in i:
6 print x
```

EXECUTION

```
sh-4.3$ python program19.py
john
Jerry
1
2
Paris
London
```

PROGRAM 21

```
1 # program to explain nested loops-If structure within
while structure
2
3 password = ""
4 while password != "secret":
5 password = raw_input("Please enter the password: ")
6 if password == "secret":
7 print("Thank you.You have entered the correct password")
8 else:
9 print("Sorry the value entered in incorrect-try again")
```

EXECUTION 1

```
sh-4.3$ python program20.py
Please enter the password: secret
Thank you. You have entered the correct password
```

EXECUTION 2

```
Please enter the password: hello
Sorry the value entered in incorrect - try again
```

EXECUTION 3

```
Please enter the password: secret
Thank you. You have entered the correct password
```

3.4 PYTHON BREAK, CONTINUE AND PASS STATEMENTS

For loops and while loops in Python allows the programmers to automate and repeat some tasks. In some situations there is a need to exit the loop completely when an external condition is triggered or there may also be a situation to skip a part of the code and start next execution. Python provides break, continue and pass statements to handle these situations.

3.4.1 Break Statement

The break statement in Python terminates the current loop and resumes execution at the next statement. It's just like the traditional break found in C.

Syntax:

```
    break;
```

1. Break statement inside for loop

```
for var in <sequence>:
        #code inside for loop
        if condition:
                break;
        #code inside for loop
#codes outside for loop
```

2. Break statement inside while loop

```
while test condition:
        #code inside for loop
        if condition:
                break;
        #code inside for loop
#code outside while loop
```

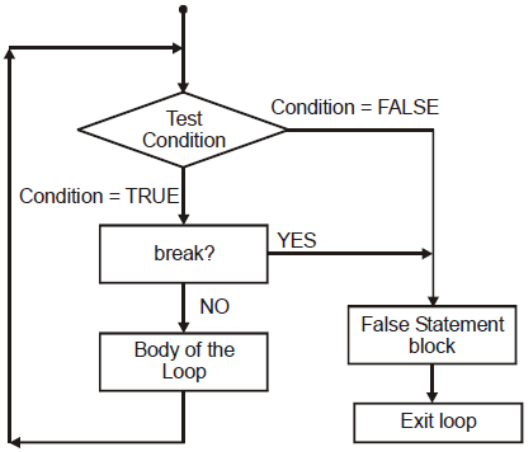

Figure 3.9 Flowchart for break Statement

PROGRAM 21

Example program for break Statement

```
1 #program to explain break statement
2 chocolates = 10
3 while (chocolates > 0):
4 print "We have ", chocolates,"number of chocolates"
5 chocolates=chocolates-1
6 if chocolates == 5:
7 break
8 print('Alarm!!! Chocolate thief')
9 input('Press enter to exit')
```

EXECUTION

```
sh-4.3$ python program21.py
We have 10 number of chocolates
We have 9 number of chocolates
We have 8 number of chocolates
We have 7 number of chocolates
We have 6 number of chocolates
Alarm!!! Chocolate thief
Press enter to exit
```

3.4.2 Continue Statement

The continue statement moves the control to the beginning of the while or for loop. The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.

Syntax:

```
continue;
```

1. Continue statement inside for loop

```
for var in <sequence>:
        #code inside for loop
        if condition:
                continue;
        #code inside for loop
#code outside for loop
```

2. Continue statement inside while loop

```
while test condition
        #code inside while loop
        if condition:
                continue
        #code inside while loop
#code outside while loop
```

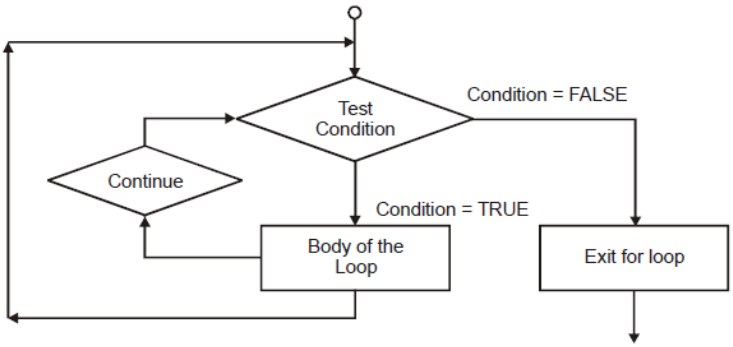

Figure 3.10 Flowchart for continue statement

PROGRAM 22

Example program for continue Statement

```
1 #program to explain continue statement
2
3 apples = 0
4 oranges = 0
5 while (oranges != -1):
6 apples = input("Enter number of apples = ")
7 oranges = input("Enter number of oranges = ")
8 if (oranges == 0):
9 continue
10 print "Total fruits in the basket =", apples + oranges
```

EXECUTION 1

```
sh-4.3$ python program22.py
Enter number of apples = 2
Enter number of oranges = 3
Total fruits in the basket = 5
```

Enter number of apples = 2
Enter number of oranges = 0
Enter number of apples = 4
Enter number of oranges = 1
Total fruits in the basket = 5

EXECUTION 2

Enter number of apples = 2
Enter number of oranges = -1
Total fruits in the basket = 1

3.4.3 Pass Statement

The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute. The pass statement is a null operation; nothing happens when it executes.

Syntax:

```
pass
```

Pass statement inside for loop

```
for var in <sequence>:
        #code inside for loop
                if condition:
                        pass;
        #code inside for loop
#code outside for loop
```

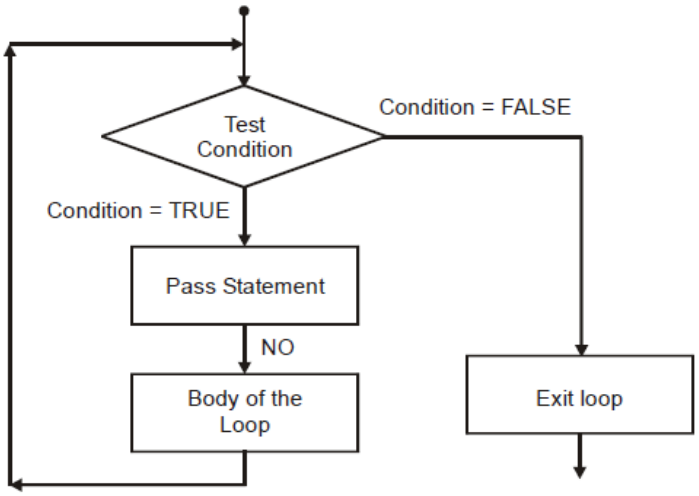

Figure 3.11 Flowchart for pass Statement

PROGRAM 23

Example program for pass Statement

1 #program to explain pass statement

```
2
3 for letter in 'hello':
4 if letter == 'o':
5 pass
6 print 'This is pass block'
7 print 'Current Letter :', letter
8 print "Thank you!"
```

EXECUTION

```
sh-4.3$ python program23.py
Current Letter : h
Current Letter : e
Current Letter : l
Current Letter : l
This is pass block
Current Letter : o
Thank you!
```

3.5 FRUITFUL FUNCTIONS

Functions that return values are called as fruitful functions. In many programming languages, a function that doesn't return any value is called a procedure.

The return statement is followed by an expression which is evaluated. Its result is returned to the caller as the "fruit" of calling this function.

Figure 3.12 Fruitful Function

Simple Example for Fruitful Function

Let's create a very simple mathematical function called add. The add function takes two numbers as parameters and return the result of adding two numbers.

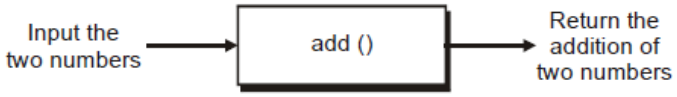

Figure 3.13 Function to add two numbers

PROGRAM 24

Simple Example for fruitful function

```
1 # Fruitful function with return statement
2
3 def add(x,y):
4 sum = x + y
```

```
5 return sum
6
7 a = input("Enter a = ")
8 b = input("Enter b = ")
9 print "Addition of two numbers = ", add(a,b)
```

EXECUTION

```
sh-4.3$ python program24.py
Enter a = 2
Enter b = 5
Addition of two numbers = 7
```

Sometimes it is useful to have multiple return statements, one in each branch of a conditional statement. As soon as a return statement executes, the function terminates without executing any subsequent statements. Code that appears after a return statement is called dead code.

PROGRAM 25

```
1 # Fruitful function with multiple return statements
2
3 def original(x):
4 if x < 0:
5 print "This is if"
6 return -x
7 else:
8 return x
9 print "This is else"#Dead code
10
11 a = input("Enter value = ")
12 print original(a)
```

EXECUTION

```
sh-4.3$ python program25.py
Enter value = 9
9
Enter value = -8
This is if
8
```

3.5.1 Parameters in Fruitful Functions

A function in Python

* Takes input data, called parameters or arguments
* Performs some computations
* Returns the result

A function definition which is already defined in Unit II is:

```
def func(param1, param2):
    #computations
    return result
```

Once a function is defined, it can be called from the main program or from another function.

Functions call statement syntax

```
result = function_name(parameter1, parameter2)
```

Parameter is the input data that is sent from one function to another. The parameters are of two types:

i) Formal parameters

* This parameter defined as part of the function definition.
* The actual parameter value is received by the formal parameter.

ii) Actual parameters

* This parameter defined in the function call.

PROGRAM 26

```
1 # Function with actual and formal parameters
2
3 def cube(x): # x is the formal parameter
4 return x*x*x
5
6 a = input("Enter the number = ")
7 b = cube(a) # a is the actual parameter
8 print "cube of the given number =", b
```

EXECUTION

```
sh-4.3$ python program26.py
Enter the number = 2
cube of the given number = 8
```

In the above example, the program takes the input value from input statement using statement number 6. cube(a) is the function name and a is the actual parameter as described in statement number 7. This invokes the function definition at statement number 3.

The parameter in the function definition is called the formal parameter. The value of a is collected by formal parameter x. The return statement at statement number 4, compute the cube of the given number and return the value, which is collected by variable b at statement number 7.

3.5.2 Parameter Passing Techniques

There are two types of parameter passing in programming languages.

1. Call by value

In call by value, a copy of actual arguments is passed to formal arguments and any changes made to the formal arguments have no effect on the actual arguments.

2. Call by reference

In call by reference, the address of actual arguments is passed to formal arguments. By accessing the addresses of formal arguments, it will be reflected in the actual arguments too.

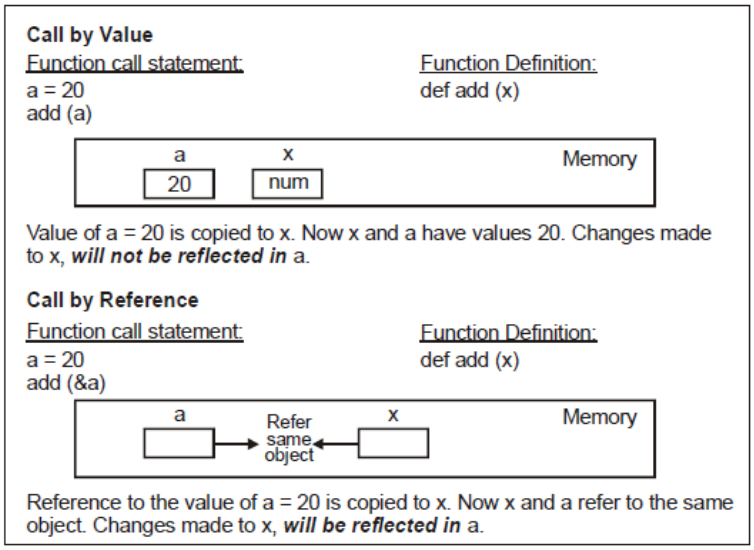

Figure 3.14 Call by value and Call by reference

But Python is neither "call-by-value" nor "call-by-reference". Python uses a mechanism, which is known as "Call-by-Object", also called "Call by Object Reference" or "Call by Sharing". If arguments like integers, strings or tuples are passed to a function, the passing acts like call-by-value.

PROGRAM 27

```
1 #Python program - call by value example
2 k = 2
3
4 def sqre(n): # n = 2, value is copied
5 n = n * n # n = 2 * 2 = 4
6 square = n # square = 4
7 return square
8
9 j = sqre(k) # value of k = 2 is passed to the function sqre
10 print "Square of the given number =", j
11 print k # k value = 2
```

EXECUTION

```
sh-4.3$ python program27.py
Square of the given number = 4
2
```

PROGRAM 28

```
1 #Python program - call by reference example
```

```
2
3 def change_list(the_list):
4 print 'Inside the Function =', the_list
5 the_list.append('raj')
6 print 'New list inside function = ', the_list
7
8
9 my_list = ['john', 'jack', 'atlee']
10
11 print 'My list before function call =', my_list
12 change_list(my_list)
13 print 'My list after function call =', my_list
#call by reference
```

EXECUTION

```
sh-4.3$ python program28.py
My list before function call = ['john', 'jack', 'atlee']
Inside the Function = ['john', 'jack', 'atlee']
New list inside function = ['john', 'jack', 'atlee', 'raj']
My list after function call = ['john', 'jack', 'atlee', 'raj']
```

3.5.3 Void Functions

It is possible to compose a function without a return statement. Functions like this are called void functions and they return none.

PROGRAM 29

```
1   # Void Function
2   def simple() #Function definition without return statement
3   print 'Hello'
4   print 'This is an example for void function'
5   simple() #Function call statement
```

EXECUTION

```
sh-4.3$ python program29.py
Hello This is an example for void function
```

3.5.4 Scope of the Variable

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

```
s = 10 <------- Here, s is the Global variable
def f1():
        s= 55 <------- Here, s is the Local variable
        print s          # output = 55
```

PROGRAM 30

```
1 # Scope of the variable in the functions
2
3 def f():
4 s = "Inside function!"
5 print(s)
6
7 s = "Outside function!"
8 f()
9 print(s)
```

EXECUTION

```
sh-4.3$ python program30.py
Inside function! Outside function!
```

3.5.5 Composition

When a function is called from within another function, it's called composition. If this type of nested function is used, the inner function has its scope only in the outer function, so it is most often useful when the inner function is being returned or when it is being passed into another function.

Syntax:

```
def outer();
def inner(a);
        return a
        return inner
...
f = outer()
f
```

PROGRAM 31

```
1 # Composition of functions
2
3 def make_adder(x):
4 def add(y):
5 print "Inside inner", x + y
6 return add
7
8 plus5 = make_adder(5)
9 print (plus5(12))
```

EXECUTION

```
sh-4.3$ python program31.py
17
```

3.5.6 Python Recursive Function

In Python, a function is recursive if it calls itself and has a termination condition. Termination condition stops the function from calling itself.

Limitations of recursions

Every time a function calls itself and stores some memory. Thus, a recursive function could hold much more memory than a traditional function.

Example - Factorial function using Recursion

The mathematical definition of factorial is: n! = n * (n-1)!,
Example: 3! = 3 x 2 x 1 = 6.

PROGRAM 32

```
1 # Recursive functions
2
3 def factorial(n):
4 if n == 0:
5 return 1
6 else:
7 return n * factorial(n - 1)
8
9 num = input("Enter the input = ")
10 print "Factorial of the given number =", factorial(num)
```

EXECUTION

```
sh-4.3$ python program32.py
Enter the input = 5
Factorial of the given number = 120
```

3.5.7 Merits of using Functions in a Program

* Dividing a large program into functions allow the programmers to read and debug easily.
* A function allows reusing code instead of rewriting it.
* A function allows testing small parts of program in isolation from the rest.

3.6 STRINGS

Strings are amongst the most popular types in Python. A string is a sequence of characters ie. They can be letter, a number, or a backslash. Python strings are "immutable" which means they cannot be changed after they are created. The various built-in methods are described below.

3.6.1 String Built-in Methods

Method name	*Description*	*Example*
Create	Strings are created by enclosing characters within single quotes or double quotes or triple quotes.	str = "Hello World
Accessing the characters	Square brackets [ ] are used to access characters in a string. Positive index - start counting from front Negative index - start counting from last	str = "Hello World" print str[5] print str[-5]
Slicing Strings	Extracting chunk of characters in a string str[start:end:stetpsize]	print str[3:10:2]
Length	It returns the length of the string.	print len(str)
Count	Count how many times, the particular character is available in the string	print str.count('l') print s.count(' ')
Find	Find the location of the particular character in the string	print str.find("H")
Index	Find the starting location of the substring	print str.index("World")
Concatenate strings	To concatenate strings in Python use the "+" operator.	print str1+str2
Join(str)	Add : between every char in the string	print ":".join(str)
ljust(width, [fillchar])	Padding is done using the specified fillchar for the length 'width'	print str1.ljust(50, '0')
lower()	Converts all uppercase letters to lowercase.	print str.lower()

Method name	*Description*	*Example*
upper()	Converts lowercase letters to uppercase.	print str.upper()
lstrip()	Removes all leading whitespace in string.	print str. lstrip()
rstrip()	Removes all trailing whitespace of string.	print str. rstrip()
strip()	Performs both lstrip() and rstrip() on string	print str. strip()
max(str)	Returns the max alphabetical character from the string str.	print max(str)
min(str)	Returns the min alphabetical character from the string str.	print min(str)
replace(old, new,[max])	Replaces all occurrences of old in string with new or at most max occurrences if max given.	print str.replace ("java", "Cent OS")
startswith(str)	Determines if string starts with substring str; returns true if so and false otherwise.	print str.endswith('str')
endswith(str)	Determines if string ends with substring str; returns true if so and false otherwise.	print str.endswith('str')
title()	Returns title cased version of string, ie. all words begins with uppercase and the rest are lowercase.	print str. title()
swapcase()	Change case for all letters in string	print str.swapcase ()

Method name	*Description*	*Example*
capitalize	Return the string with it's first character capitalized and the rest lowercased. If the first character is a space, the space is unchanged, the rest lowercased	print string.capitalize()
split(separator)	Split or breakup a string and add to a string array using a defined separator. If no separator is defined then whitespace will be used by default	print string.split(" ")
Repeat Strings	Repeat the string n times	print str * 3
Reverse the string	Reverse the characters in the string	print ''.join(reversed(str))
isalnum()	Returns true if string has at least 1 character and all characters are alphanumeric or false otherwise.	print str. isalnum()
isalpha()	Returns true if string has at least 1 character and all characters are alphabetic or false otherwise.	print str. isalpha()
isdigit()	Returns true if string contains only digits and false otherwise	print str. isdigit()
islower()	Returns true if string has lowercase characters and false otherwise	print str. islower()
isupper()	Returns true if string has uppercase characters and false otherwise	print str. isupper()
isnumeric()	Returns true if string contains only numeric characters and false otherwise.	print str. isnumeric()
isspace()	Returns true if string contains only whitespace characters and false otherwise.	print str. isspace()
istitle()	Returns true if string is properly "titlecased" and false otherwise.	print str. istitle()

3.6.2 Example Program for String Manipulations

PROGRAM 33

```
1 # String Manipulations
2
3 str = "Python programming "
4 str1 = "reg_no:892898!"
5 print "Character at location 3 = ", str[2]
6 print "Length of the string = ", len(str)
```

```
7 print "Number of times the given character (m) = ",str.count('m')
8 print "Number of times , blank space = ",str.count(' ')
9 print "Location of the character (p) = ",str.find("p")
10 print "Index of the sub string = ", str.index("program")
11 print "Join the character : with each character of the string = ", ":".join(str)
12 print "Join the empty space with each character of the string = "," ".join(str)
13 print "Concatenation of two strings using + symbol =", str + str1
14 print "String in lower case = ", str.lower()
15 print "String in upper case = ",str.upper()
16 print "Title case of the String = ",str.title()
17 print "Swap the case of the String = ",str.swapcase()
18 print "Capiltalize the String reg no. 892898 = ",str1.capitalize()
19 print "Reverse the String = ",".join(reversed(str))
20 print "Split the String on white space =",str.split(" ")
21 print "Split the String on character m ",str.split("m")
22 print "Replace old string with new string = ", str.replace("Python", "Cent OS")
23 print "Replace old string with new string = ", str.replace("o","Cent OS")
24 print "Replace old with new string only 1 time = ",str. replace("o","Cent OS", 1)
25 print "Maximum characters in the string reg No.892898 = ", Maxstr1)
26 print "Minimum characters in the string reg No.892898 = ", min(str1)
27 print "Padding the string by $ symbol",str1.ljust(25, '$')
28 print "&" * 10 #prints character &, 10 times
29 print str * 3 #prints str 3 times
30
31 str2 = " Django programming "
32 print "Removes all leading whitespace in string = ",str2.lstrip()
33 print "Removes all trailing whitespace in string = ",str2.rstrip()
34 print "Removes all whitespace characters in string = ",str2.strip()
35
36 #String Slicing
37 print "Get first character of the string = ", str1[0]
38 print "Get only 1 character = ", str1[0:1]
39 print "Get first 3 characters in the string = ", str1[0:3]
40 print "Get last three characters = ", str1[-3:]
41 print "Get first 3 characters & last 3 characters in the string = ",str1[0:3] + str1[-3:]
42 print "Get the three characters from location 3 till location 7 = ",str1[3:10]
43 print "Return a character by moving forward 2 positions = ",str1[3:10:2]
44 print "Get all characters from 3rd place = ", str1[3:]
45 print "Get all characters except last 3 characters =", str1[:-3]
46
47 str1 ="DB124"
```

```
48 print str1
49 print "Check alphanumeric characters =", .str1.isalnum()
50 print "Check all are alphabetic characters =",str1.isalpha() #false, if space
51 print "Check if string fully digits =", str1.isdigit()
52 print "Check for title words =", str1.istitle()
53 print "Check for uppercase characters =", str1.isupper()
54 print "Check for lowercase characters =", str1.islower()
55 print "Check for whitespace characters =", str1.isspace()
56 print "Check if string ends with character B =",str1.endswith('B')
57 print "Check if string ends with character B =",str1.startswith('B')
```

EXECUTION

```
sh-4.3$ python program33.py
Character at location 3 = t
Length of the string = 19
Number of times the given character (m) = 2
Number of times , blank space = 2
Location of the character (p) = 7
Index of the sub string = 7
Join the character : with each character of the string = P:y:t:h:o:n: :p:r:o:g:r:a:m:m:i:n:g:
Join the empty space with each character of the string = P y t h on p r o g r a m m i n g
Concatenation of two strings using + symbol = Python .. programming reg_no:892898!
String in lower case = python programming
String in upper case = PYTHON PROGRAMMING
Title case of the String = Python Programming
Swap the case of the String = pYTHON PROGRAMMING
Capitalize the String reg no. 892898 = Reg_no:892898!
Reverse the String = gnimmargorp nohtyP
Split the String on white space = ['Python', 'programming', '']
Split the String on character m = ['Python progra', '', 'ing ']
Replace old string with new string = Cent OS programming
Replace old string with new string = PythCent OSn prCent OSgramming
Replace old with new string only 1 time = PythCent OSn programming
Maximum characters in the string reg No. 892898 = r
Minimum characters in the string reg No. 892898 = !
Padding the string by $ symbol reg_no:892898!$$$$$$$$$$$&&&&&&&&&&&
Python programming Python programming Python programming
Removes all leading whitespace in string = Django programming
Removes all trailing whitespace in string = Django programming
Removes all whitespace characters in string = Django programming
Get first character of the string = r
Get only 1 character = r
```

Get first 3 characters in the string = reg
Get last three characters = 98!
Get first 3 characters & last 3 characters in the string = reg98!
Get the three characters from location 3 till location 7 = _no:892
Return a character by moving forward 2 positions = _o82
Get all characters from 3rd place = _no:892898!
Get all characters except last 3 characters = reg_no:8928 DB124
Check alphanumeric characters = True
Check all are alphabetic characters = False
Check if string fully digits = False
Check for title words = False
Check for uppercase characters = True
Check for lowercase characters = False
Check for whitespace characters = False
Check if string ends with character D = False
Check if string ends with character D = True

3.6.3 The String Module

The string module provides additional tools to manipulate strings. This module contains a number of functions to process standard Python strings. In recent versions, string built-in functions are available as string module functions.

PROGRAM 34

```
1 # String Module
2
3 import string
4
5 text = "This is python programming"
6
7 print "upper", "=>", string.upper(text)
8 print "lower", "=>", string.lower(text)
9 print "split", "=>", string.split(text)
10 print "join", "=>", string.join(string.split(text), "+")
11 print "replace", "=>", string.replace(text, "python", "Java")
12 print "find", "=>", string.find(text, "python"),string.find(text, "Java")
13 print "count", "=>", string.count(text, "n")
```

EXECUTION

```
sh-4.3$ python program34.py
upper => THIS IS PYTHON PROGRAMMING
lower => this is python programming
split => ['This', 'is', 'python', 'programming']
join => This+is+python+programming
replace => This is Java programming
```

find => 8 -1
count => 2

3.7 PYTHON LISTS

One of the most fundamental data structures in any language is the array. Python doesn't have a native array data structure, but it has the list. List is one of the compound data type available in Python often referred to as sequences. In Python programming, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas. Items in a list need not be of the same data type. A list is mutable; it means the contents of the list are changed.

list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5];
list3 = ["a", "b", "c", "d"]

Index operator [] is used to access an item in a list. Index starts from 0. Python allows negative indexing for its sequences. The index of -1 refers to the last item. Nested list are accessed using nested indexing.

PROGRAM 35

```
1 #Simple program for list
2 my_list = ['p','r','o','b','e']
3 print "First item in the List = ", (my_list[0])
4 print "Third item in the List = ",(my_list[2])
5 print "Last item in the List = ",(my_list[-1])
6
7 # Nested List
8 n_list = ["Happy", [2,0,1,5], ['john',78,17.3,"hi"]]
9 print "2nd item in list 1 = ", n_list[0][1]
10 print "3rd item in list 3 = ", n_list[2][3]
```

EXECUTION

```
sh-4.3$ python program35.py
First item in the List = p
Third item in the List = o
Last item in the List = e
2nd item in list 1 = a
3rd item in list 3 = hi
```

3.7.1 Basic List Operations

Lists respond to the + and * operators much like strings; they mean concatenation and repetition which results is a new list.

S.No	Method name	Description	Syntax
1.	Length	Find the length of the list	len(List)
2.	Concatenation	Concatenate two lists	L1+L2
3.	Repetition	Repeat the item multiple times	['Hi!'] * 4
4.	Append	Add an element to the end of the list	list.append(item)
5.	Insert	Insert an item at the defined index	list.insert(index, item)
6.	Extend	Add all elements of a list to the another list	list.extend(new list)
7.	Remove	Removes an item from the list	list.remove(item)
8.	Pop	Removes an element at the given index	list.pop()
9.	Clear	Removes all items from the list	list.clear()
10.	Index	Returns the index of the first matched item	list[index]
11.	Count	Count the number of times	list.count(item)
12.	Sort	Sort items in a list in ascending order	list.sort()
13.	Reverse	Reverse the order of items in the list	list.reverse()

Table 3.1 Basic List Operations

3.7.2 Example Program for Basic Operations in Lists

PROGRAM 36

```
1 # Basic List operations
2 L1 = [56,78,98,78]
3 print "Items in the list = ", L1
4 L1.insert(2,"python")
5 print "Inserted item in the list at location 2 = ", L1
6 print "Length of items in the list = ", len(L1)
7 L2 = ['h','e','l','l','o']
```

```
8 L3 = L1+L2
9 print "Concatenated items in the list = ", L3
10 print "Repeat item 4 times =", ['Hi!'] * 4
11 L4 = ["hi","john","jack"]
12 L3.extend(L4)
13 print "Extended items in the list =", L3
14 del L3[4]
15 print "Delete the item in the list at location 4 = ", L3
16 del L3[1:5]
17 print "Delete the items from location 1 to 5 = ", L3
18 L3.remove('l')
19 print "Remove an item = ", L3
```

EXECUTION

```
sh-4.3$ python program36.py
Items in the list = [56, 78, 98, 78]
Inserted item in the list at location 2 = [56, 78, 'python', 98, 78]
Length of items in the list = 5
Concatenated items in the list = [56, 78, 'python', 98, 78, 'h', 'e', 'l', 'l', 'o']
Repeat item 4 times = ['Hi!', 'Hi!', 'Hi!', 'Hi!']
Extended items in the list = [56, 78, 'python', 98, 78, 'h', 'e', 'l', 'l', 'o', 'hi', 'john', 'jack']
Delete the item in the list at location 4 = [56, 78, 'python', 98, 'h', 'e', 'l', 'l', 'o', 'hi', 'john', 'jack']
Delete the items from location 1 to 5=[56, 'e', 'l', 'l', 'o', 'hi', 'john', 'jack']
Remove an item = [56, 'e', 'l', 'o', 'hi', 'john', 'jack']
```

3.7.3 Example Program for Built-in Functions in Lists

PROGRAM 37

```
1   # Built-in functions in lists
2   L = [56,78,98,78]
3   print "Items in the list = ", L
4   L.reverse()
5   print "Reversed items in the list = ", L
6   L.append(568)
7   print "Append item in the list = ", L
8   L.insert(0, "hi")
9   print "Insert new item after 1st item = ", L
10  print "Number of items 78 is in the list = ", L.count(78)
11  print "Length of items in the list = ", len(L)
12  L.sort()
13  print "Sort the items in the list = ", L
14  print "Minimum element in the list = ", min(L)
15  print "Maximum element in the list = ", max(L)
```

```
16  print "Pop last time in the list = ", L.pop()
17  print "Pop 1st time in the list = ",L.pop(0)
```

EXECUTION

```
sh-4.3$ python program37.py
Items in the list = [56, 78, 98, 78]
Reversed items in the list = [78, 98, 78, 56]
Append item in the list = [78, 98, 78, 56, 568]
Insert new item after 1st item = ['hi', 78, 98, 78, 56, 568]
Number of items 78 is in the list = 2
Length of items in the list = 6
Sort the items in the list = [56, 78, 78, 98, 568, 'hi']
Minimum element in the list = 56
Maximum element in the list = hi
Pop last time in the list = hi
Pop 1st time in the list = 56
```

3.8 ILLUSTRATIVE PROGRAMS

3.8.1 Program to Find the Square Root of the Given Number

PROGRAM 38

```
1   # Program to find the square root of the given number
2
3   import math # This will import math module
4
5   print "Square of 49 = ", math.sqrt(49)
6   num = input("Enter the number :")
7   print "Square root of the given number = ", int(math.sqrt(num))
```

EXECUTION

```
sh-4.3$ python program38.py
Square of 49 = 7.0
Enter the number :49
Square root of the given number = 7
```

3.8.2 Program to Find GCD of Two Numbers

PROGRAM 39

```
1   #GCD of two numbers
2   #Divisors of 54 are: {1,2,3,6,9,18,27,54}
3   #Divisors of 24 are: {1,2,3,4,6,8,12,24}
4   #Common divisors of 54 and 24: {1,2,3,6}
5   #Greatest common divisor of 54 and 24 = 6
6
```

```
7   d1=int(input("Enter a number = "))
8   d2=int(input("Enter another number = "))
9   rem=d1%d2
10  while rem!=0 :
11  d1=d2
12  d2=rem
13  rem=d1%d2
14  print "GCD of given numbers is = ", d2
```

EXECUTION

```
sh-4.3$ python program39.py
Enter a number = 54
Enter another number = 24
GCD of given numbers is = 6
```

3.8.3 Program to Find Exponentiation of a Given Number

```
1 # Exponentiation of a given number
2 import math
3
4 print "Exponentiation of a negative number = ", math.exp(-45.17)
5 print "Exponentiation of a negative number = ", math.exp(10)
6 print "Exponentiation of a pi = ", math.exp(math.pi)
```

EXECUTION

```
sh-4.3$ python program40.py
Exponentiation of a negative number = 2.41500621326e-20
Exponentiation of a negative number = 22026.4657948
Exponentiation of a pi = 23.1406926328
```

3.8.4 Program to Explain Enumerate() Function

PROGRAM 41

```
1 # Enumerate() function is used to iterate through a list
2 # while keeping track of the list items indices.
3
4 fruits = ['apples', 'bananas', 'blueberries', 'oranges', 'mangos']
5
6 for index, fruit in enumerate(fruits):
7 print("The fruit, " +fruit + ", is in position " + str(index) + ".")
```

EXECUTION

```
sh-4.3$ python program41.py
The fruit, apples, is in position 0.
The fruit, bananas, is in position 1.
The fruit, blueberries, is in position 2.
```

The fruit, oranges, is in position 3.
The fruit, mangos, is in position 4.

3.8.5 Program to Find Sum of Elements in the List

PROGRAM 42

```
1 #Sum of elements in the list
2 my_data = [89,221,8,23]
3 sum = 0
4 for i in my_data:
5 sum += i
6 print "Sum of elements in the list = ", sum
```

EXECUTION 1

```
sh-4.3$ python program42.py
Sum of elements in the list = 341
```

```
1 #Sum of elements in the list
2 my_data = [8,23]
3 print "Sum of elements in the list = ", sum(my_data)
```

EXECUTION 2

```
sh-4.3$ python program42.py
Sum of elements in the list = 31
```

```
1 #Adding two lists
2 list1 = [1,2,3,4,5]
3 list2 = [10,20,30,40,50]
4 sum_list1_list2 = list1 + list2
5 print "Sum of two lists = ", sum(sum_list1_list2)
```

EXECUT ION 3

```
sh-4.3$ python program42.py
Sum of two lists = 165
```

```
1 #Adding two lists
2 list1 = [1,2,3,4,5]
3 list2 = [10,20,30,40,50]
4 list3 = [(x + y) for x, y in zip(list1, list2)]
5 print "Sum of two lists = ", list3
```

EXECUTION 4

```
sh-4.3$ python program42.py
Sum of two lists = [11, 22, 33, 44, 55]
```

3.8.6 Simple Calculator Program to Add, Subtract, Multiply and Divide using Functions

PROGRAM 43

```
1 # Simple calculator to add, subtract, multiply and divide using functions
2
```

```
3 # define functions
4 def add(x, y):
5 return x + y
6
7 def subtract(x, y):
8 return x - y
9
10 def multiply(x, y):
11 return x * y
12
13 def divide(x, y):
14 return x / y
15
16 print "Select operation."
17 print "1. Add"
18 print "2. Subtract"
19 print "3. Multiply"
20 print "4. Divide"
21
22 choice = raw_input("Enter choice(1/2/3/4):")
23
24 num1 = input("Enter first number: ")
25 num2 = input("Enter second number: ")
26
27 if choice == '1':
28 print num1,"+",num2,"=", add(num1,num2)
29
30 elif choice == '2':
31 print num1,"-",num2,"=", subtract(num1,num2)
32
33 elif choice == '3':
34 print num1,"*",num2,"=", multiply(num1,num2)
35
36 elif choice == '4':
37 print num1,"/",num2,"=", divide(num1,num2)
38 else:
39 print "Invalid input"
```

EXECUTION

```
sh-4.3$ python program43.py
Select operation.
1. Add
```

2. Subtract
3. Multiply
4. Divide
Enter choice(1/2/3/4):1
Enter first number: 45
Enter second number: 54
45 + 54 = 99
Select operation.
1. Add
2. Subtract
3. Multiply
4. Divide
Enter choice(1/2/3/4):3
Enter first number: 45
Enter second number: 5
45 * 5 = 225

3.8.7 Linear Search is used to Find an Item in an Unordered list

PROGRAM 44

```
1 # Linear search is used to find an item in a unordered list.
2 # To search an item, start at the beginning of the list and continue searching
3 # until either the end of the list is reached or the item is found.
4
5 my_data = [89,45,9,21,34] #items in the list, array element start at location 0
6 num = input("Enter search number = ")
7 for i in range(0,len(my_data)):
8 if num == my_data[i]: #if item at position i
9 print "Item is location at the position = " , i
```

EXECUTION

```
sh-4.3$ python program44.py
Enter search number = 34
Item is location at the position = 4
```

3.8.8 Binary Search

Binary search is a fastest algorithm for searching an element in a sorted array. Running time of binary search is log2 N time, but the searching time of linear search is N/2. Binary search is suitable for large lists and is more efficient sorting algorithm.

The element in the array is described as below and the element to be searched is 45.

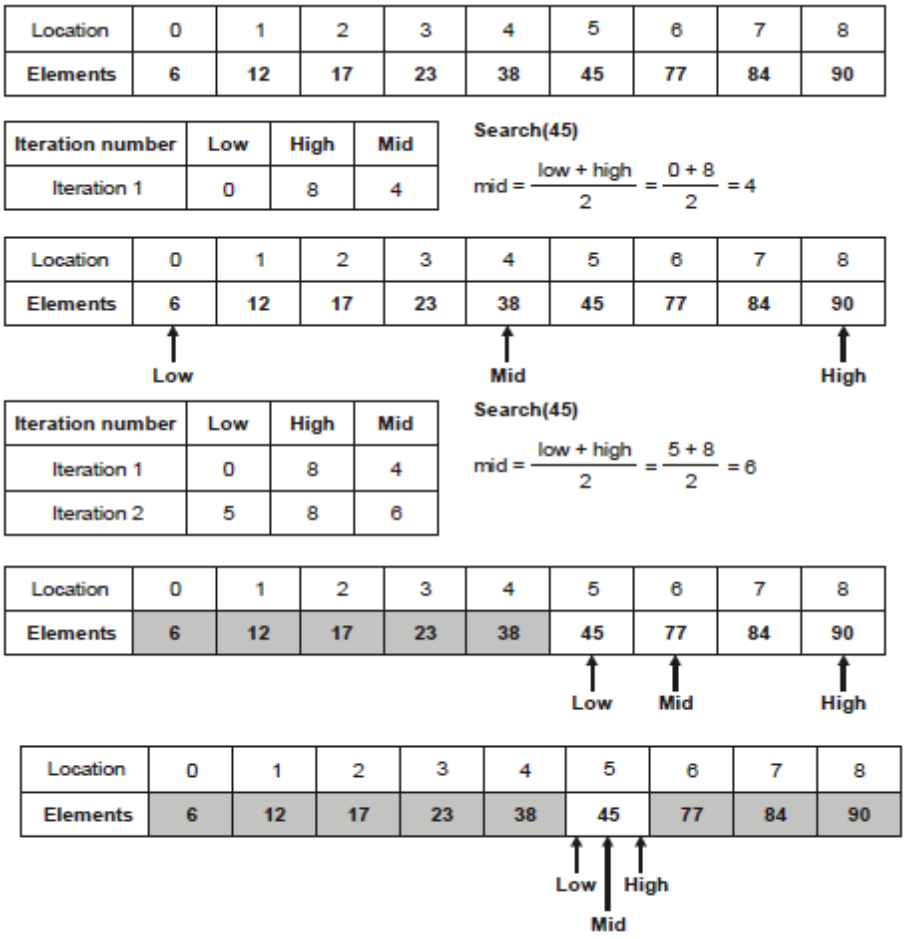

Figure 3.15 Successful Search

1. Binary search is used to find an item

PROGRAM 45

```
1 # Binary search is used to find an item in a ordered list.
2 def binary_search(item_list,item):
3 first = 0
4 last = len(item_list)-1
5 found = False
6 while( first<=last and not found):
7 mid = (first + last)//2
8 if item_list[mid] == item :
9 found = True
10 print "Element found at location =", mid
11 else:
12 if item < item_list[mid]:
13 last = mid - 1
14 else:
```

```
15 first = mid + 1
16 return found
17
18 print "Search the element 45 in the list 1:"
19 print(binary_search([6,12,17,23,38,45,77,84,90],45))
20 print "Search the element 55 in the list 2:"
21 print(binary_search([16,12,17,23,38,45,77,84,90], 5))
```

EXECUTION

```
sh-4.3$ python program45.py
Search the element 45 in the list 1:
Element found at location = 5
True
Search the element 55 in the list 2:
False Item is location at the position = 4
```

TWO MARKS QUESTIONS WITH ANSWERS

1. Define boolean datatype.

   The Boolean data type is a data type, having two values (usually denoted true and false). It is named after George Boole, who first defined an algebraic system of logic in the mid 19thcentury.

2. What are the Conditional statements used in Python?

   * if statements
   * if...else statements
   * if...elif...else statements(or) chained conditionals
   * Nested if statements

3. Define if...else statements with its syntax.

   The if..else statement evaluates test condition and executes the true statement block only when test condition is True.If the condition is False, false statement block is executed. Indentation is used to separate the blocks.

   Syntax:

   ```
   if condition:
   True Statement Block
   else:
   False Statement Block
   ```

4. Write the syntax for ternary operator.

   The syntax for ternary operaor is,

   Syntax 1:

   ```
   value_if_true_(if_condition) else value_if_false
   ```

   Syntax 2:

(value_if_true, value_if_false) [if_condition]

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

6. Write the syntax for if…elif…else conditionals.
   Syntax:

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

7. Define iteration.
   Iteration is otherwise called as looping. There are situations when programmers need to execute a block of code several number of times. Repeated execution of a set of statements is called iteration or looping.

8. What are the different iterative statements?
   Python programming language provides following types of iterative statements.
   * The for loop
   * The while statement
   * Nested loops

9. Define range() function and its syntax.
   Sequence of numbers are also generated using range() function.
   Syntax:

```
range(start_element, stop_element, step size)
```

   Default step size is equal to 1 if not provided.

10. Define the While loop.
    The while loop in Python iterates over a block of statements as long as the test condition is true.The statement body is entered only when the test condition is true. The process continues until the test condition evaluates to False.
    Syntax:

```
while condition:
```

statement_1
...
statement_n

11. Write the syntax for Nested for loops and Nested while loop statements.
Python programming language allows using one loop inside another loop.
Syntax:
Nested for loop
for iterating_var in sequence:
statement(s)
statement(s)
Syntax:
Nested while loop
while expression:
while expression:
statement(s)
statement(s)

12. What is Python Break statement?
The break statement in Python terminates the current loop and resumes execution at the next statement. It's just like the traditional break found in C.
Syntax:
break;

13. What is Python Continue statement?
The continue statement moves the control to the beginning of the while or for loop. The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.
Syntax:
continue;

14. Define Fruitful function in Python.
A function in Python
* Takes input data, called parameters or arguments.
* Performs some computations.
* Returns the result.

15. What are types of parameters in Functions?
Parameter is the input data that is sent from one function to another. The parameters are of two types:
* Formal parameters
* This parameter defined as part of the function definition.

* The actual parameter value is received by the formal parameter.
* Actual parameters
* This parameter defined in the function call.

16. What are thevarious Parameter Passing Techniques?
    There are two types of parameter passing in programming languages.
    Call by value: In call by value, a copy of actual arguments is passed to formal arguments and any changes made to the formal arguments have no effect on the actual arguments.
    Call by reference: In call by reference, the address of actual arguments is passed to formal arguments. By accessing the addresses of formal arguments, it will be reflected in the actual arguments too.

17. Write the Scope of the Variable.
    A variable in the program can be either local variable or global variable. A global variable is a variable that is declared in the main program while a local variable is a variable that is declared within the function.
    s = 10 Here, s is the Global variable
    def f1():
    s= 55 Here, s is the Local variable
    print s # output = 55

18. What is Recursive Function and its limitations?
    In Python, a function is recursive if it calls itself and has a termination condition.
    Limitations of recursions
    Every time a function calls itself and stores some memory. Thus, a recursive function could hold much more memory than a traditional function.

19. Write the merits of using Functions in a program.
    * Dividing a large program into functions allow the programmers to read and debug easily
    * A function allows reusing code instead of rewriting it.
    * A function allows testing small parts of program in isolation from the rest.

20. Write the syntax for Composition.
    When a function is called from within another function, it's called composition.
    Syntax:

```
def outer();
def inner(a);
return a
return inner
```

21. Define strings and name some methods.

A string is a sequence of characters ie. they can be letter, a number, or a backslash. Python strings are "immutable" which means they cannot be changed after they are created.

22. List some of the methods in List Operations.
    Length, Concatenation, Repetition, Append, Insert, Extend, Remove, Pop, Clear, Index, Count, Sort and Reverse.

23. State the differences between Linear search and Binary search.
    Linear search, also known as the sequential search is the simplest search algorithm. It searches for a specified value in a list by checking every element in the list. Binary search is also a method used to locate a specified value in a sorted list. Binary search method halves the number of elements checked thereby reducing the time taken.

REVIEW QUESTIONS

1. Explain If structure in detail. Explain with suitable examples.
2. Explain iteration structure in detail with suitable examples.
3. Write a Python program that accepts a string and calculate the number of digits and letters.
   Given String: Python 3.2
   Expected Output:
   Number of letters : 6
   Number of digits: 2
4. Write a Python program to display the current date and time.
5. Explain Python Break, Continue and Pass Statements with examples.
6. Write a Python program that prints all the numbers from 0 to 6 except 3 and 6.
7. What is a fruitful function? What are the parameters used in the fruitful functions?
8. What are the two parameter passing techniques? Explain them with examples.
9. Write a Python function that accepts a string and calculate the number of upper case letters and lower case letters.
   Sample String :Nothing WORTH comes EASY.
   Expected Output :
   No. of Upper case characters :10
   No. of Lower case Characters : 11