UNIT III CONTROL FLOW, FUNCTIONS

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

3.1 BOOLEAN VALUES

Any object can be tested for truth value, for use in an if or while condition or as operand of the Boolean operations below. The following values are considered false:

- None
- False
- zero of any numeric type, for example, 0, 0L, 0.0, 0j.
- any empty sequence, for example, '', (), [].
- any empty mapping, for example, {}.

All other values are considered true — so objects of many types are always true.

Operations and built-in functions that have a Boolean result always return 0 or False for false and 1 or True for true, unless otherwise stated.

3.2 OPERATORS

Operators are the constructs which can manipulate the value of operands.

Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and + is called operator.

Types of Operator

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators

- Membership Operators
- Identity Operators

3.2.1 Arithmetic Operators

Assume variable a holds 10 and variable b holds 20, then −

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity):	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

Example.py

```
a = 21 b
= 10 c =
0 c = a +
b
print "Line 1 - Value of c is ", c

c = a - b
print "Line 2 - Value of c is ", c

c = a * b
print "Line 3 - Value of c is ", c

c = a / b
print "Line 4 - Value of c is ", c

c = a % b
print "Line 5 - Value of c is ", c
```

```
a = 2 b
= 3 c =
a**b
print "Line 6 - Value of c is ", c
a = 10 b
= 5 c =
a//b
print "Line 7 - Value of c is ", c
```

Output:

```
Line 1 - Value of c is 31
Line 2 - Value of c is 11
Line 3 - Value of c is 210
Line 4 - Value of c is 2
Line 5 - Value of c is 1
Line 6 - Value of c is 8
Line 7 - Value of c is 2
```

3.2.2 Comparison Operators

These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.

Assume variable a holds 10 and variable b holds 20, then –

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Example.py

```
a = 21 b
= 10 c =
0


if ( a == b ):    print "Line 1 - a is
equal to b" else:    print "Line 1 - a
is not equal to b"

if ( a != b ):
  print "Line 2 - a is not equal to b"
else:    print "Line 2 - a is equal to
b"

if ( a <> b ):
  print "Line 3 - a is not equal to b"
else:    print "Line 3 - a is equal to
b"

if ( a < b ):
  print "Line 4 - a is less than b"
else:    print "Line 4 - a is not less
than b"

if ( a > b ):
  print "Line 5 - a is greater than b"
else:    print "Line 5 - a is not greater
than b"

a = 5; b =
20; if ( a <=
b ):
  print "Line 6 - a is either less than or equal to  b"
else:    print "Line 6 - a is neither less than nor equal
to  b"

if ( b >= a ):
  print "Line 7 - b is either greater than  or equal to b"
else:    print "Line 7 - b is neither greater than  nor equal
to b"
```

Output

```
Line 1 - a is not equal to b
Line 2 - a is not equal to b
Line 3 - a is not equal to b
```

Line 4 - a is not less than b
Line 5 - a is greater than b
Line 6 - a is either less than or equal to b
Line 7 - b is either greater than or equal to b

3.2.3 Assignment Operators

Assume variable a holds 10 and variable b holds 20, then –

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Example.py

```
a = 21 b =
10 c = 0

c = a + b print "Line 1 - Value
of c is ", c

c += a print "Line 2 - Value of
c is ", c

c *= a print "Line 3 - Value of
c is ", c
```

```
c /= a  print "Line 4 - Value of
c is ", c

c  = 2 c %= a print "Line 5 -
Value of c is ", c

c **= a print "Line 6 - Value of
c is ", c

c //= a print "Line 7 - Value of
c is ", c
```

Output:

```
Line 1 - Value of c is 31
Line 2 - Value of c is 52
Line 3 - Value of c is 1092
Line 4 - Value of c is 52
Line 5 - Value of c is 2
Line 6 - Value of c is 2097152 Line
7 - Value of c is 99864
```

3.2.4 Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. Assume if a = 60; and

b = 13; Now in binary format they will be as follows − a = 0011 1100 b = 0000 1101

---

a&b = 0000 1100

a|b = 0011 1101 a^b

= 0011 0001

~a = 1100 0011

There are following Bitwise operators supported by Python language

Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)

~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

Example.py

```
a = 60          # 60 = 0011 1100  b
= 13          # 13 = 0000 1101  c =
0

c = a & b;       # 12 = 0000 1100
print "Line 1 - Value of c is ", c

c = a | b;       # 61 = 0011 1101
print "Line 2 - Value of c is ", c

c = a ^ b;       # 49 = 0011 0001
print "Line 3 - Value of c is ", c

c = ~a;          # -61 = 1100 0011
print "Line 4 - Value of c is ", c

c = a << 2;      # 240 = 1111 0000
print "Line 5 - Value of c is ", c

c = a >> 2;      # 15 = 0000 1111 print
"Line 6 - Value of c is ", c
```

Output:

```
Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240 Line
6 - Value of c is 15
```

3.2.5 Logical Operators

Operator	Description	Example

and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are nonzero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

3.2.6 Membership Operators

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Example.py a
= 10
b = 20
list = [1, 2, 3, 4, 5];

if (a in list):
  print "Line 1 - a is available in the given list" else:
  print "Line 1 - a is not available in the given list"

if (b not in list):
  print "Line 2 - b is not available in the given list" else:
  print "Line 2 - b is available in the given list"

a = 2 if (a in
list):
  print "Line 3 - a is available in the given list"
else: print "Line 3 - a is not available in the
given list"

Output:

Line 1 - a is not available in the given list
Line 2 - b is not available in the given list
Line 3 - a is available in the given list

3.2.7 Identity Operators

Identity operators compare the memory locations of two objects. There are two Identity operators explained below:

Operator	Description	Example
Is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id(x) is not equal to id(y).

Example.py

```
a        =
20 b =
20

if ( a is b ):
  print "Line 1 - a and b have same identity" else:
  print "Line 1 - a and b do not have same identity"

if ( id(a) == id(b) ):
  print "Line 2 - a and b have same identity" else:
  print "Line 2 - a and b do not have same identity"

b        =
30 if ( a
is b ):
  print "Line 3 - a and b have same identity" else:
  print "Line 3 - a and b do not have same identity"

if ( a is not b ):
  print "Line 4 - a and b do not have same identity" else:
  print "Line 4 - a and b have same identity"
```

Output:

```
Line 1 - a and b have same identity
Line 2 - a and b have same identity
Line 3 - a and b do not have same identity
Line 4 - a and b do not have same identity
```

3.2.8 OPERATORS PRECEDENCE

The following table lists all operators from highest precedence to lowest.

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Example.py

```
a = 20
b = 10
c = 15
d = 5 e
= 0

e = (a + b) * c / d      #( 30 * 15 ) / 5
print "Value of (a + b) * c / d is ",  e

e = ((a + b) * c) / d     # (30 * 15 ) / 5
print "Value of ((a + b) * c) / d is ",  e

e = (a + b) * (c / d);    # (30) * (15/5) print
"Value of (a + b) * (c / d) is ",  e

e = a + (b * c) / d;      #  20 + (150/5)
print "Value of a + (b * c) / d is ",  e
```

Output:

```
Value of (a + b) * c / d is 90
```

Value of ((a + b) * c) / d is 90
Value of (a + b) * (c / d) is 90
Value of a + (b * c) / d is 50

3.3 CONDITIONAL STATEMENTS

1. if statement
2. if-else statement
3. if-elif-else statement
4. Nested if / conditional statement
5. Chained conditional

3.3.1 if statement

- Used to check a condition and control the flow of execution **Syntax:**

```
if expression:
    statements
```

- Expression is a logical expression which tests and results either true or false

Flowchart:

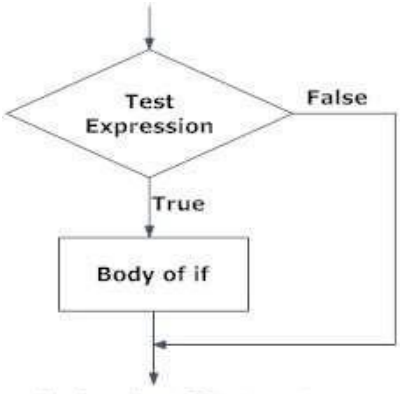


Fig: Operation of if statement

Example.py number=10 if number>0:
print(number," is a positive number")
number=-1 if number>0:
  print(number," is a positive number")

Output:
10 is a positive number

3.3.2 if-else statement

- If there are two statements to be executed alternatively, then if-else statement is used.
- If the condition is true, then true statements are executed otherwise statements of else part is executed.

Syntax:

```
if expression:
        statements
else:
        statements
```

Flowchart:

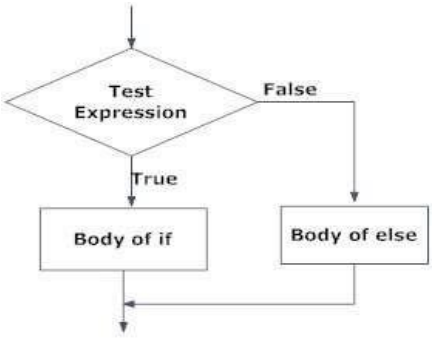


Fig: Operation of if...else statement

example.py

```
number=28 if
number%2==0:
  print(number," is a even number") else:
  print(number," is a odd number")
```

Output:

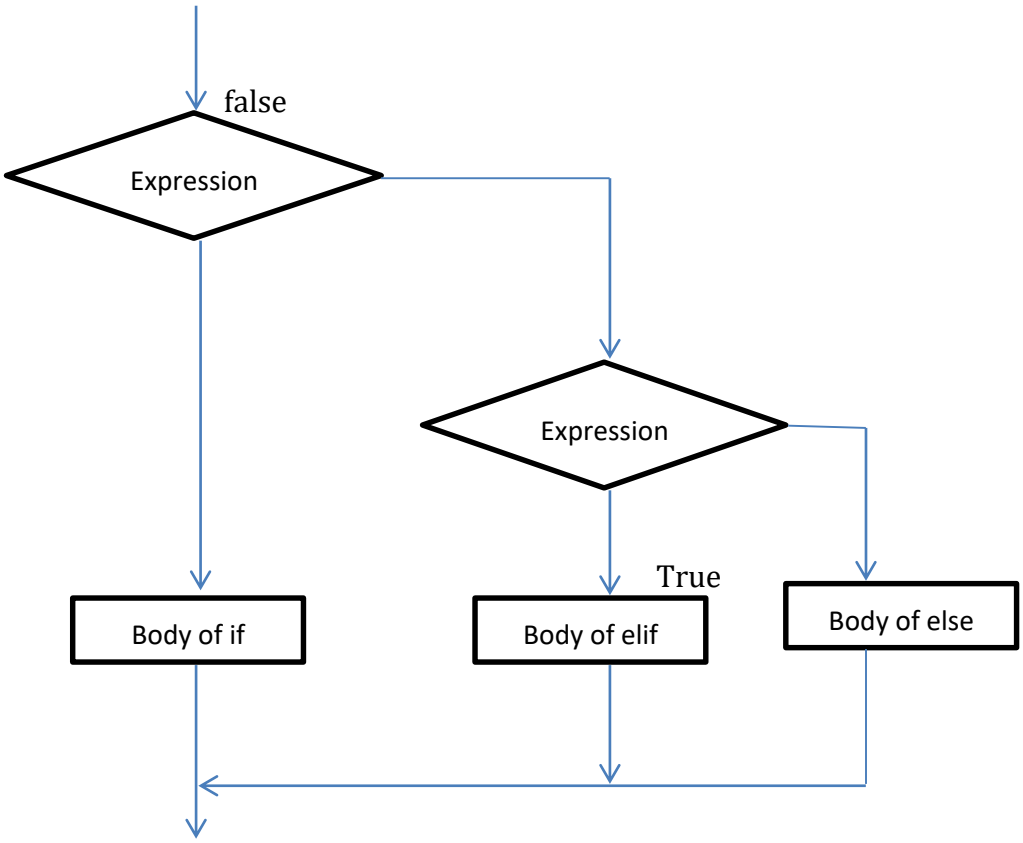
28 is a even number

3.3.3 if-elif-else statement

- It is used to test more than one condition.
- If there are more than two alternatives to select, then nested if statements are used. **Syntax:**

```
if expression:
        body of if elif
expression:
        body of elif
--
--
else:
        body of else
```

Flowchart:



Example.py a=float(input("Enter the number:"))
if a>0: print("The given number is a positive number") elif a==0:
print("The given number is Zero") else:
print("The given number is negative number")

Output:

Enter the number:-1
The given number is negative number
>>>
Enter the number:0
The given number is Zero
>>>
Enter the number:100
The given number is a positive number

3.3.4 Nested if statement

- Here a if-elif-else construct is placed inside another if-elif-else construct.
- Indentation has to be clear in flow of statements.

Example.py

#Program to print a number in text format from 1 to 5
value=int(input("Enter the value:")) if value<=5: if value==1: print("One") elif value==2:

```
print("Two")    elif value==3:       print("Three")
elif value==4:       print("Four")    elif value==5:
print("Five") else:
   print("try a number inside the limit")
```

Output:

```
>>>Enter the value:4
Four
>>>Enter the value:99 try a
number inside the limit
```

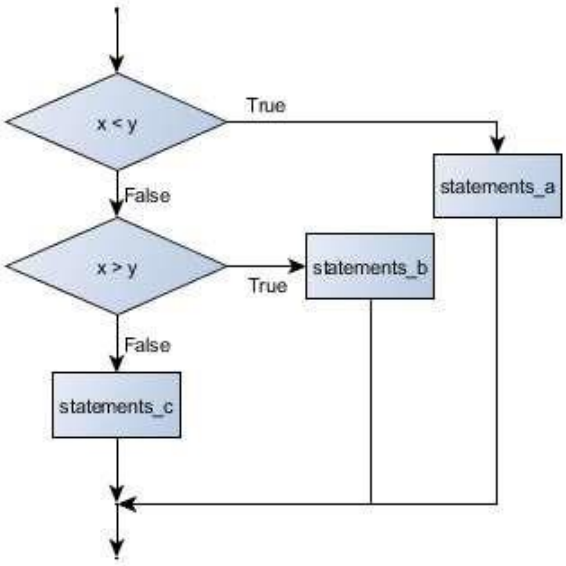
3.3.5 Chained Conditional

- When there are more than two possibilities it requires more than two branches.

Syntax:

```
if expression:
        body of if
 elif expression:
        body of elif
else:
       body of else
```

Flowchart:



Example.py

```
value=10 if
value==20:
   print("Got a true value ",value) elif
 value==15:
```

```
    print("Got a true value ",value) elif
value==10:
    print("Got a true value ",value) else:
    print("Got a false value ",value)
print("Program Over")
```

Output:

```
>>>Got a true value  10
Program Over
```

3.4 ITERATION

- Repeated execution of set of statements until a specified condition becomes true is called as iteration.
- Types:
  1. while
  2. for

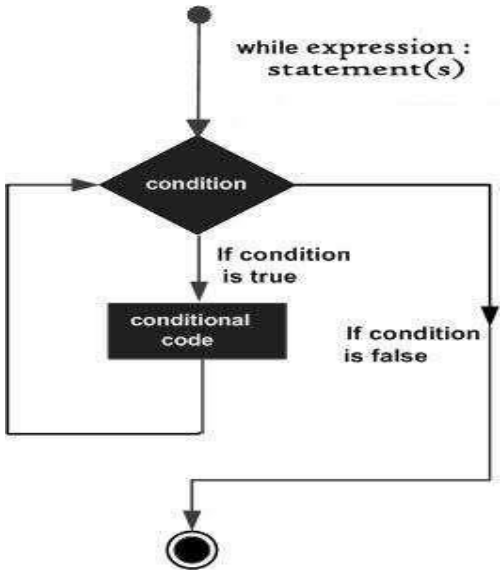
3.4.1 while loop

- It is an entry controlled loop
- It executes a block of code repeatedly till the condition becomes true.

Syntax:

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

Flowchart:



Example.py

```
count=0
while(count<5):
   print("the count value is",count)
count=count+1
```

Output:

```
the count value is 0 the
count value is 1 the
count value is 2 the
count value is 3 the
count value is 4
```

3.4.2 Nested while

- It is when a while loop is placed inside another while loop.

Syntax:

```
while expression:
    statements
    while expression:
        statements
```

Example.py

```
x=-2 y=2 while
x<=y:
print("x=",x)
x=x+1
while(x<=0):
    print("x is negative")
x=x+1
```

Output:

```
x= -2 x is
negative x is
negative x=
1 x= 2
```

3.4.3 for loop

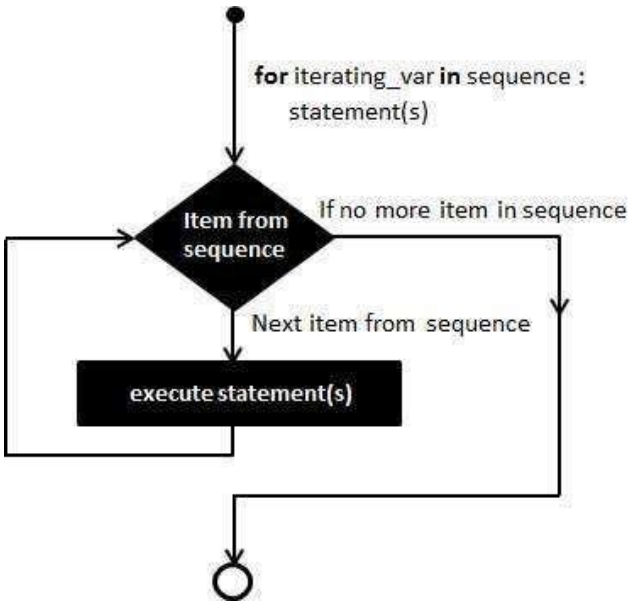
- Executes a sequence of statements multiple times.

Syntax: for val in sequence:

Body of for

- Val is a variable that takes value of the item inside the sequence on each iteration.
- Loop continues until the last item in the sequence is reached.
- For loop contains initialization & condition part, but increment and decrement need not to be defined in python.

Flowchart:



Example.py

```
word = "computer"
for letter in word:
print(letter)
```

Output:

```
c o
m
```

p u

t e

r

example.py

```
num=[20,30,40] for
i in num:
   print("The value of i is",i)
```

Output:

The value of i is 20

The value of i is 30 The

value of i is 40

3.4.4 Nested for loops

- For loop inside another for loop is called as nested for loop.

Syntax:

```
                for val in sequence:
    for val in sequence:
        statements     statements
```

Example.py

```
for x in range(1,3):    for
y in range(1,3):
print(x,"*",y,"=",x*y)
```

Output:

1 * 1 = 1

1 * 2 = 2

2 * 1 = 2

2 * 2 = 4

3.4.5 range()

- This function is used for iteration using for loop
- Default initial value of range() is 0. ➢ This returns a list of values.

Syntax:

```
range([start],stop,[step])
```

where, start is starting value stop is end value,which stop-1

and step is difference between each number in that sequence.

Example.py

```
print("Range with one argument") for
i in range(5):
   print(i)
print("Range with two argument") for
i in range(10,20):
   print(i)
print("Range with three argument") for
i in range(50,70,3):
   print(i)
```

Output:

```
Range with one argument
0
1
2
3
4
Range with two argument
10
11
12
13
14
15
16
17
```

18
19
Range with three argument
50
53
56
59
62
65
68

xrange()

- This works similar to range() but returns a xrange object.
- This is used when huge billion amount of values are to be generated.
- This works only in Pyhton 2v.
- It uses less memory.

Syntax:

```
xrange([start],stop,[stop])
```

example.py

```
print("xrange with single argument") for
i in xrange(6):
   print(i)
print("xrange with two arguments") for
i in xrange(1,7):
   print(i)
print("xrange with three arguments") for
i in xrange(1,10,3):
   print(i)
```

Output:

xrange with single argument

```
0
1
2
3
4
5       xrange with two
arguments
1
2
3
4
5
6       xrange with three
arguments
1
4
7
```

3.5 LOOP CONTROL STATEMENTS

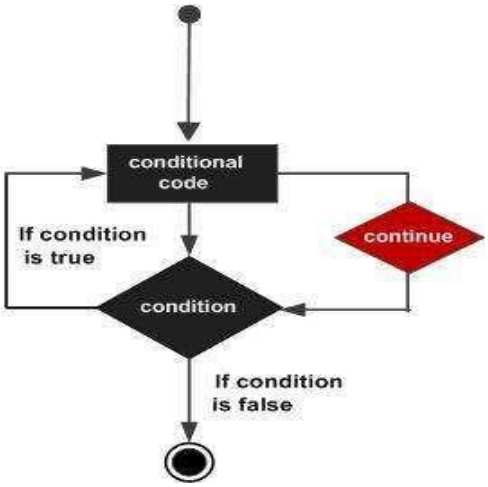
3.5.1 Continue Statement

- This returns control to the beginning of the loop.
- It rejects all remaining statements in the current iteration of the loop and moves the control back to the top of the loop.
- It can be used both in while and for loop.

Syntax:

```
continue
```

Flowchart:



Example.py

```
for x in range(1,10):
   if x==5:
      continue
print("x=",x)
```

Output:

x= 1 x=

2 x= 3

x= 4 x=

6 x= 7

x= 8 x=

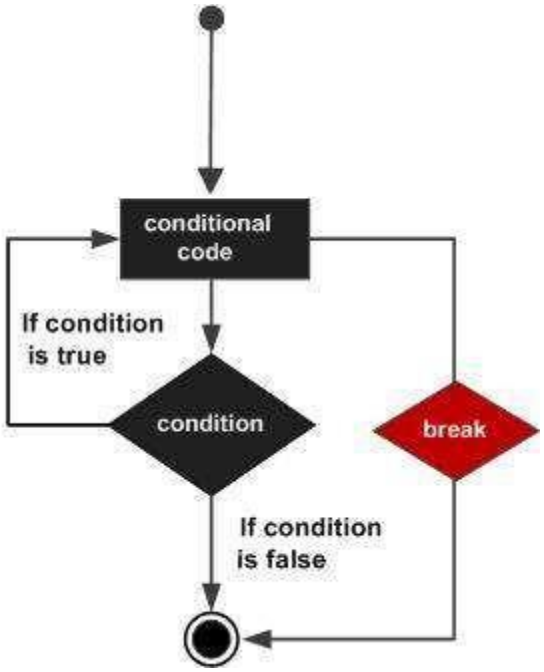
9

3.5.2 break

- It brings control out of the loop.
- Terminates the loop statement and transfers execution to the statement immediately following the loop.

Syntax:

Flowchart:



Example.py

```
for x in range(1,10):
if x==5:        break
print("x=",x)
```

Output:

```
x= 1 x=
2 x= 3
x= 4
```

3.5.3 pass

- The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.
- Used to write empty loops.

Syntax:

```
pass
```

Example.py

```
for letter in "python":
if letter=="h":
    pass
    print("pass block")
print("letter=",letter)
```

Output:

```
letter= p
letter= y
letter= t pass
block letter=
h letter= o
letter= n
```

3.6 FRUITFUL FUNCTIONS

- Functions that return values are called as fruitful functions.
- The opposite is void function.
- The return statement is followed by an expression which is evaluated.
- Its result is returned to the caller as the “fruit” of calling function.

Example.py

```
def square(b):
```

```
  c=b*b    return c b=5
result=square(b) print("The square
of ",b," is ",result)
```

Output:

```
The square of  5  is  25
>>>
```

3.6.1 Return values

- Return is a keyword which is used to return a value from the function definition block to the function calling line.
- Return takes zero, values or an expression.
- Default value is none.
- If many values are used then, use a tuple or list.

3.6.2 Parameters / Arguments (Refer Unit 2 for detailed explanation) ➢

These are the inputs given to a function.

- Input values (parameters) are passed from function calling line to function definition block, where here it is called as arguments.
- Types:
    1. Required arguments.
    2. Keyword arguments.
    3. Default arguments.
    4. Variable-length arguments.

#Example for Required argument

Example.py

```
def a(s):    print(s)
return a(100) a(100,2)
```

Output:

```
100
TypeError: a() takes 1 positional argument but 2 were given
>>>
```

#Example for Keyword argument

Example.py

```
def a(s,d):    print(s,d)
return
a(s=100,d="Black") a(d="Black",s=100)
```

Output:

```
100 Black
```

100 Black

#Example for Default argument
Example.py

```
def a(s,d=20):
print(d,s)
return a(15)
a(15,5)
```

Output:

```
20 15
5 15
```

#Example for Variable-length argument
Example.py

```
def a(*s):    for i in s:
print(i)    return
a(10,20,30)    a("hai")
```

Output:

```
10
20 30
hai
```

#Program to add two numbers by passing parameters
Example.py

```
def add(a,b):    c=a+b    return c
y=add(2,2) print(y)
```

Output:

```
4
>>>
```

3.6.3 Scope of Variables

- The part of a program where a variable is accessible is called its scope, and the duration for which the variable exists is called its lifetime.
- There are two basic scopes of variables.
    1. Global variables
    2. Local variables
- A variable defined in the main body of a program at the top is called a global variable and is visible throughout the life.
- A variable defined inside a function is local to that function.

Example.py def ss(a): a=10
print("value inside function:",a)
return a a=20 print("value when return
is used:",ss(a)) print("value outside
function:",a) **Output:**
value inside function: 10 value
when return is used: 10 value
outside function: 20

3.6.4 Composition

- The ability of calling one function from within another function is called as composition.

Example.py
#Program to find square of a number
c=float(input("Enter a value:"))
d=c**2 print(d)

Output:
Enter a value:2.5
6.25

- Here the input function is used inside float function.
- Input function considers any input value as a string, and according to our needs we have to use either int() or float() to convert the input value into integer or floating value to process in our program.

3.6.5 Recursion

- Recursion is a process of calling a function by itself again and again until some condition is satisfied.
- A recursive function must have
    1. A statement to test whether the function is calling itself again.
    2. A statement that calls the function itself must be an argument.
    3. A conditional statement 4. A return statement.

Advantages:

1. The code look clean and elegant.
2. Large problems broken down into small problems Disadvantages:

1. Logic is hard to follow. 2.

It takes more memory.

3. It is hard to debug.

Example.py

#Program to find factorial using recursion

```
def factorial(n):
if n==1:
return 1    else:
    return n*factorial(n-1) print(factorial(5))
```

Output:

```
120
>>>
```

Example.py

#Program to find Fibonacci series till 10 terms

```
def fibo(n):
if n<=1:
return n
else:
    return(fibo(n-1)+fibo(n-2))
terms=10 print("Fibonacci
series") for i in range(terms):
print(fibo(i))
```

Output:

```
Fibonacci series
0
1
1
2
3
5
8
13
21
34
```

3.7 STRINGS

- It is a sequence of characters treated as a single data item enclosed within either single or double quotes or even triple quotes.

Example1.py

```
example='I am found
of chocolates' print(example)
example="I am found of ice creams"
print(example) example='''I am
found of cakes too''' print(example)
example='' print(example)
```

Output:

```
I am found of chocolates
I am found of ice creams
I am found of cakes too
```

Example2.py

```
#string with single quote
var1='python' print(var1)
#string with double quote
var1="python" print(var1)
#string with triple quote
var1="""Python is one of the programming
    language""" print(var1)
```

Output:

```
python python
Python is one of the programming
    Language
```

3.7.1 String Slices

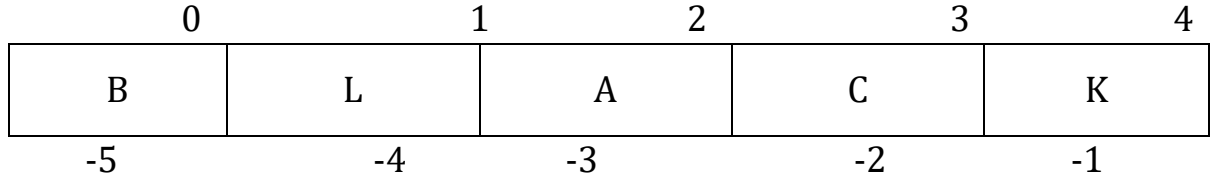
- A portion or a filtered part of a string is called as slice.
- Slice is used to take sub parts of either from list or string.

Syntax:

```
Substring=originalstring[start:end]
```

Start- start index value which is included
End- end index value which is excluded

Example



- The above example contains both positive and negative indexing.

Example1.py

```
b='Live and let live'
print(b[0:5])
print(b[3:])
print(b[:6]) print(b[:])
```

Output:

```
Live  e and
let live Live
a
Live and let live
>>>
```

Example2.py

```
str='Python Programming' print("The Given String is",str)
print("The first character in the String is",str[0])
print("The character starting from 3rd to 5th position is",str[2:5])
print("The String starting from 3rd character is",str[2:])
print("The last character of the string is",str[-1]) print("The String from negative indices is",str[-5:])
```

Output:

The Given String is Python Programming

The first character in the String is P

The character starting from 3rd to 5th position is tho

The String starting from 3rd character is thon Programming

The last character of the string is g

The String from negative indices is mming

3.7.2 Immutability

- Strings in python are immutable i,e after created it cannot be changed.
- No assignments can be done later but updation is possible.

Example1.py

#NO ITEM ASSIGNMENT IN PYTHON

var1="python" var1[2]="T"

Output:

Traceback (most recent call last):

var1[2]="T"

TypeError: 'str' object does not support item assignment

Example2.py

#NO ITEM DELETION IN PYTHON

var1="python" del var1

Output:

Traceback (most recent call last):

TypeError: 'str' object does not support item deletion

Example3.py

var1="python" del

var1 print(var1)

Output:

Traceback (most recent call last): print(var1)

NameError: name 'var1' is not defined

Example4.py

v1 = 'Are you okay!' print("Updated
String:",v1[:13] +' baby')

Output:

Updated String: Are you okay! Baby

3.7.3 String Operators

Assume string variable **a** holds 'Hello' and variable **b** holds ' boys ', then –

Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b will give Helloboys
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give -HelloHello
[]	Slice - Gives the character from the given index	a[1] will give e
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give ell
in	Membership - Returns true if a character exists in the given string	H in a will give true
not in	Membership - Returns true if a character does not exist in the given string	M not in a will give true

Example.py

```
a='I hate bitter food' b="badly" print("The Concatenation of
the String is:",a+b) print("The Repetition of the String
is:",a*2) print("The Slicing of the string is:",a[2]) print("The
Slicing of the string is:",a[3:6]) print("The membership
operator of the string is:",'t' in a) print("The membership
operator of the string is:",'B' not in b)
```

Output:

```
The Concatenation of the String is: I hate bitter foodbadly
The Repetition of the String is: I hate bitter foodI hate bitter food
The Slicing of the string is: h
The Slicing of the string is: ate
The membership operator of the string is: True
The membership operator of the string is: True
>>>
```

3.7.4 String formatting Operator ➢

Symbol used is %.

➢ It is mainly used for print()

Format Symbol	Conversion
%c	character
%s	string conversion via str() prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer (lowercase letters)
%X	hexadecimal integer (UPPERcase letters)
%e	exponential notation (with lowercase 'e')
%E	exponential notation (with UPPERcase 'E')
%f	floating point real number
%g	the shorter of %f and %e
%G	the shorter of %f and %E

3.7.5 Escape sequences in string

Escape Sequence	Description
\newline	Backslash and newline ignored
\\	Backslash
\'	Single quote
\"	Double quote
\a	ASCII Bell
\b	ASCII Backspace
\f	ASCII Formfeed
\n	ASCII Linefeed

\r	ASCII Carriage Return
\t	ASCII Horizontal Tab
\v	ASCII Vertical Tab
\ooo	Character with octal value ooo
\xHH	Character with hexadecimal value HH

3.7.6 String functions & String Methods

➢ Many built-in string methods of strings are available. ➢
Some examples are.,

capitalize()	center()	casefold()	count()	endswith()	encode()
find()	format()	index()	split()	rsplit()	title()
zfill()	isalpha()	isdecimal()	isdigit()	islower()	isupper()
join()	strip()	partition()	replace()	startswith()	isnumeric()

1. capitalize()function

- This returns a string with first letter in capital letters.
- **Syntax**:

  stringname.capitalize ()

- This does not takes any argument.
- If first letter is already a capital letter or non-alphabet, then it returns the original string.

Example.py

```
var1="python
programming"
print("The result of capitalize function is",var1.capitalize())
```

Output:

The result of capitalize function is Python programming

2.center() function

- This method returns a string padded with specified character to fill.
- It does not modify the original string.
- **Syntax**:

stringname.center(width[,fillchar])

where width is length of the string & fill char is character to be filled.

Example.py

str="I lost my books!" print(str.center(18,'a'))

Output:

aI lost my books!a

3.casefold() function

- This method removes all case variations in a string.
- It is used for caseless matching.
- It doesnot take any parameters.
- **Syntax:**

string.casefold()

Example.py a1="PYTHON

programming"

print("The result of casefold function is",a1.casefold()) **Output:**

The result of casefold function is python programming

>>>

4.count() function

- This function returns number of occurrences of substring in the range[start,end].
- **Syntax**

Stringname.count(substring,start= … , end=…) where

substring whose count is to be found.& start,end or optional.

Example.py var1="Engineering Knowledge" print("The result of

count function is",var1.count('e'))

Output:

The result of count function is 4

5.endswith() function

- This method returns True if a string ends with the specified suffix otherwise return false. ➢ **Syntax**:

Str.endswith(suffix[,start[,end]])

where suffix is string to be checked * start, end are optional.

Example.py t="English is just a language
not knowledge" result=t.endswith("
knowledge") print(result)
result=t.endswith(" knowledge.")
print(result) result=t.endswith("not")
print(result)

Output:
True
False
False
>>>

6.find() function

- This function is used to return the lowest index of the substring (if found) otherwise returns -1.
- **Syntax**:

str.find(sub[,start[,end]])

Example.py var1="PYTHON
programming"
print("The result of find function is",var1.find('PYTHON')) print("The
result of find function is",var1.find('Pyhton'))

Output:
The result of find function is 0
The result of find function is -1
>>>

7.format()function

- This is used to make the presentation of output more neatly ➢ **Syntax**:

  template.format(posarg1, posarg2,.....,keyarg0=v0,keyarg1=v1,....)

  posarg is positional arguments & keyarg is keyword argument. **Example.py**

```
print("Hi {},welcome to {}".format("all","python"))
#positional arguments print("Hi {1},welcome to
{0}".format("all","python"))
#keyword arguments print("Hi {key},welcome to
{name}".format(key="all",name="python"))
#mixed arguments print("Hi {0},welcome to
{name}".format("all",name="python"))
```

Output:

```
Hi all,welcome to python Hi
python,welcome to all
Hi all,welcome to python
Hi all,welcome to python
>>>
```

8.index()function

- This function returns the index of a substring inside the string otherwise it raises a value error exception.
- **Syntax**:

```
str.index(sub[,start[,end]])
```

Example.py

```
var1="welcome to python
programming" result=var1.index('to
python') print("substring 'to python'
",result)
```

Output:

```
substring 'to python'  8
```

9.split() function

- It breaks up a string at the specified separator and returns.

- **Syntax**: str.split([separator[,max]])

where separator is the place where the split occurs & max is maximum number of splits.

Example.py new='abcdef' items='black,blue,green' print(new.split())

print(items.split(','))

Output:

['abcdef']

['black', 'blue', 'green']

10.startswith()function

- This method returns True if a string starts with the specified prefix otherwise returns false. ➢ **Syntax**:

str.startswith(prefix[,start[,end]])

Example.py t1="Computer
Language"
r1=t1.startswith('Language',9)
print(r1)
r1=t1.startswith('Language',2)
print(r1)

Output:
True False

3.7.7 String module

- String module has numerous predefined methods to process in python.

Example.py (String module)

import string s="Welcome to the world of Robotics"
print("Upper case:",str.upper(s)) print("Lower
case:",str.lower(s)) print("Split:",str.split(s))
print("Join:"," ".join(s))
print("Replace:",str.replace(s,"Robotics","Innovation"))

```
print("Find:",str.find(s,"world"),str.find(s,"of"))
print("Count:",str.count(s,"t"))Output:
```

Output:

Upper case: WELCOME TO THE WORLD OF ROBOTICS

Lower case: welcome to the world of robotics

Split: ['Welcome', 'to', 'the', 'world', 'of', 'Robotics']

Join: W e l c o m e t o t h e w o r l d o f R o b o t i c s

Replace: Welcome to the world of Innovation

Find: 15 21

Count: 3

Example.py (string methods without using string module functions)

```
text="Welcome to the world of Robotics"
print("Upper case:",text.upper()) print("Lower
case:",text.lower()) print("Split:",text.split())
print("Join:","+".join(text.split()))
print("Replace:",text.replace("Robotics","Innovation"))
print("Find:",text.find("world"),text.find("of")) print("Count:",text.count("t"))
```

Output:

Upper case: WELCOME TO THE WORLD OF ROBOTICS
Lower case: welcome to the world of robotics
Split: ['Welcome', 'to', 'the', 'world', 'of', 'Robotics']
Join: Welcome+to+the+world+of+Robotics
Replace: Welcome to the world of Innovation
Find: 15 21 Count:
3

3.8 LISTS AS ARRAYS

- A List is a group of values.
- Each and every value in a list is called as elements or items.
- Each element is separated using ,. ➢ List can also be sliced as [m:n].

Example.py

```
a=[1,2,5,8,9] b=[“a”,”g”,”r”]

c=[67,”red”,90,”yellow”]
```

3.8.1 Accessing values in Lists:

- [] are used to access values in a list for slicing along with the index or indices to obtain value available at that index.
- Slice[m:n] where m is starting index which is included & n in end index which is excluded. **Example.py**

```
l1=['English','Maths',1000,3000]
l2=[10,20,30,40,50] print(l1[0])
print(l2[2:4])
print(l1[:-2]) print(l2[-6])
```

Output:

```
English
[30, 40]
['English', 'Maths']
Traceback (most recent call last):
  File "python", line 6, in <module> IndexError:
list index out of range
```

3.8.2 Updating list

- Single or multiple elements can be updated in a list by giving the slice on the lefthand side of the assignment operator and also add the elements in a list with the append() method.

Example.py

```
s=['Red','green','Blue','yellow','Purple']
print("value at index 2:",s[2])
s[2]='Lavender' print(s)
s.append('Orange') print(s)
```

Output:

```
value at index 2: Blue
['Red', 'green', 'Lavender', 'yellow', 'Purple']
['Red', 'green', 'Lavender', 'yellow', 'Purple', 'Orange']
```

3.8.3 Deleting elements in a List

- When index of element is known then, 'del' keyword is used to delete.
- When element to delete is known, then remove() method is used.

Example.py

```
s=['Red','green','Blue','yellow','Purple']
print(s) del s[2] print(s)
s.remove('yellow') print(s)
```

Output:

```
['Red', 'green', 'Blue', 'yellow', 'Purple']
['Red', 'green', 'yellow', 'Purple']
['Red', 'green', 'Purple']
```

3.9 ILLUSTRATIVE PROGRAMS

1. Square root of a number

```
import math num=16
result=math.sqrt(num)
print("Square root value is:",result)
```

Output:

Square root value is: 4.0

2. GCD of two numbers

```
def gcd(a,b): if(b==0): return a
else:
   return gcd(b,a%b) a=int(input("Enter the first number:")) b=int(input("Enter the second number:")) result=gcd(a,b)
print("GCD is:",result)
```

Output:

```
Enter the first number: 30
Enter the second number: 12
GCD is: 6
```

3. Exponentiation of a number

```
def power(base,exp):
if(exp==1): return(base)
if(exp!=1):
 return(base*power(base,exp-1))
base=int(input("Enter base: "))
exp=int(input("Enter exponential value: "))
print("Result:",power(base,exp))
```

Output:

```
Enter base:  3
Enter exponential value:  2
Result: 9
```

4. Sum of array of numbers

```
b=[2,5,8,1,0,9,5] sum=0 for i in b:
  sum=sum+i print("Sum of array of numbers in
list is :",sum)
```

Output:

```
Sum of array of numbers in list is : 30
```

5. Linear search

```
list=[8,4,10,54,89] search=int(input("Enter the
number to search:")) length=len(list) for i in
range(0,length):   if list[i]==search:
   print(search," is found at the position
",i+1)    break else:
  print("Number not found")
```

Output:

```
Enter the number to search: 10
10  is found at the position  3

Enter the number to search: 1
Number not found
```

6. Binary search

```
def binarysearch(sortedlist,n,x):     start = 0     end = n - 1
while(start <= end):      mid = (start + end)//2
mid=int(mid) #gives mid a rounded value if it is in float
if (x == sortedlist[mid]):
     return mid      elif(x <
sortedlist[mid]):
     end = mid - 1

    else:
     start = mid + 1
return -1

n =5 sortedlist =
[10,23,38,41,50]
```

```
x = int(input("Enter the number to search: "))
position = binarysearch(sortedlist, n, x) if(position
!= -1):
 print("Entered number %d is present at position: %d"%(x,position+1)) else:
 print("Entered number %d is not present in the list"%x)
```

Output:

Enter the number to search: 38

Entered number 38 is present at position: 3