Standard Data Types:

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types:

- Numbers
- String
- Boolean
- List
- Tuple
- Set
- Dictionary

Python Numbers:

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

Python supports four different numerical types:

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

Python allows you to use a lowercase L with long, but it is recommended that you use only an uppercase L to avoid confusion with the number 1. Python displays long integers with an uppercase L.

A complex number consists of an ordered pair of real floating-point numbers denoted by x + yj, where x is the real part and b is the imaginary part of the complex number.

For example:

Program:

```
a = 3
b = 2.65
c = 98657412345L
d = 2+5j
print "int is",a
print "float is",b
print "long is",c
print "complex is",d
```

Output:

```
int is 3
float is 2.65
long is 98657412345
complex is (2+5j)
```

Python Strings:

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

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

Program:
```
str ="WELCOME"
print str # Prints complete string
print str[0] # Prints first character of the string
print str[2:5] # Prints characters starting from 3rd to 5th
print str[2:] # Prints string starting from 3rd character
print str * 2 # Prints string two times
print str + "CSE" # Prints concatenated string
```

Output:
```
WELCOME
W
LCO
LCOME
WELCOMEWELCOME
WELCOMECSE
```

Built-in String methods for Strings:

SNO	Method Name	Description
1	capitalize()	Capitalizes first letter of string.
2	center(width, fillchar)	Returns a space-padded string with the original string centered to a total of width columns.
3	count(str, beg= 0,end=len(string))	Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.
4	decode(encoding='UTF-8',errors='strict')	Decodes the string using the codec registered for encoding. Encoding defaults to the default string encoding.
5	encode(encoding='UTF-8',errors='strict')	Returns encoded string version of string; on error, default is to raise a Value Error unless errors is given with 'ignore' or 'replace'.
6	endswith(suffix, beg=0, end=len(string))	Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise.
7	expandtabs(tabsize=8)	Expands tabs in string to multiple spaces; defaults to 8 spaces per tab if tabsize not provided.
8	find(str, beg=0 end=len(string))	Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.

9	index(str, beg=0, end=len(string))	Same as find(), but raises an exception if str not found.
10	isalnum()	Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.
11	isalpha()	Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.
12	isdigit()	Returns true if string contains only digits and false otherwise.
13	islower()	Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.
14	isnumeric()	Returns true if a unicode string contains only numeric characters and false otherwise.
15	isspace()	Returns true if string contains only whitespace characters and false otherwise.
16	istitle()	Returns true if string is properly "titlecased" and false otherwise.
17	isupper()	Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.
18	join(seq)	Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.
19	len(string)	Returns the length of the string.
20	ljust(width[, fillchar])	Returns a space-padded string with the original string left-justified to a total of width columns.
21	lower()	Converts all uppercase letters in string to lowercase.
22	lstrip()	Removes all leading whitespace in string.
23	maketrans()	Returns a translation table to be used in translates function.
24	max(str)	Returns the max alphabetical character from the string str.
25	min(str)	Returns min alphabetical character from the string str.
26	replace(old, new [, max])	Replaces all occurrences of old in string with new or at most max occurrences if max given.
27	rfind(str, beg=0,end=len(string))	Same as find(), but search backwards in string.
28	rindex(str, beg=0, end=len(string))	Same as index(), but search backwards in string.
29	rjust(width,[, fillchar])	Returns a space-padded string with the original string right-justified to a total of width columns.
30	rstrip()	Removes all trailing whitespace of string.
31	split(str="", num=string.count(str))	Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given.
32	splitlines (num=string.count('\n'))	Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed.

33	startswith(str, beg=0,end=len(string))	Determines if string or a substring of string (if starting index beg and ending index end are given) starts with substring str; returns true if so and false otherwise.
34	strip([chars])	Performs both lstrip() and rstrip() on string.
35	swapcase()	Inverts case for all letters in string.
36	title()	Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase.
37	translate(table, deletechars="")	Translates string according to translation table str(256 chars), removing those in the del string.
38	upper()	Converts lowercase letters in string to uppercase.
39	zfill (width)	Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero).
40	isdecimal()	Returns true if a unicode string contains only decimal characters and false otherwise.

Example:

```
str1="welcome"
print "Capitalize function---",str1.capitalize()
print str1.center(15,"*")
print "length is",len(str1)
print "count function---",str1.count('e',0,len(str1))
print "endswith function---",str1.endswith('me',0,len(str1))
print "startswith function---",str1.startswith('me',0,len(str1))
print "find function---",str1.find('e',0,len(str1))
str2="welcome2017"
print "isalnum function---",str2.isalnum()
print "isalpha function---",str2.isalpha()
print "islower function---",str2.islower()
print "isupper function---",str2.isupper()
str3="          welcome"
print "lstrip function---",str3.lstrip()
str4="77777777cse777777";
print "lstrip function---",str4.lstrip('7')
print "rstrip function---",str4.rstrip('7')
print "strip function---",str4.strip('7')
str5="welcome to java"
print "replace function---",str5.replace("java","python")
```

Output:

```
Capitalize function--- Welcome
****welcome****
length is 7
count function--- 2
endswith function--- True
startswith function--- False
find function--- 1
isalnum function--- True
isalpha function--- False
```

islower function--- True
isupper function--- False
lstrip function--- welcome
lstrip function--- cse777777
rstrip function--- 77777777cse
strip function--- cse
replace function--- welcome to python

Python Boolean:

Booleans are identified by True or False.

Example:

Program:

a = True
b = False
print a
print b

Output:

True
False

Data Type Conversion:

Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type name as a function. For example, it is not possible to perform "2"+4 since one operand is integer and the other is string type. To perform this we have convert string to integer i.e., **int("2") + 4 = 6**.

There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

Function	Description
int(x [,base])	Converts x to an integer.
long(x [,base])	Converts x to a long integer.
float(x)	Converts x to a floating-point number.
complex(real [,imag])	Creates a complex number.
str(x)	Converts object x to a string representation.
repr(x)	Converts object x to an expression string.
eval(str)	Evaluates a string and returns an object.
tuple(s)	Converts s to a tuple.
list(s)	Converts s to a list.
set(s)	Converts s to a set.
dict(d)	Creates a dictionary, d must be a sequence of (key, value) tuples.
frozenset(s)	Converts s to a frozen set.
chr(x)	Converts an integer to a character.
unichr(x)	Converts an integer to a Unicode character.
ord(x)	Converts a single character to its integer value.
hex(x)	Converts an integer to a hexadecimal string.
oct(x)	Converts an integer to an octal string.

Types of Operators:

Python language supports the following types of operators.
- Arithmetic Operators   **+, -, *, /, %, **, //**
- Comparison (Relational) Operators   **= =, ! =, < >, <, >, <=, >=**
- Assignment Operators   **=, +=, -=, *=, /=, %=, **=, //=**
- Logical Operators   **and, or, not**
- Bitwise Operators   **&, |, ^, ~,<<, >>**
- Membership Operators   **in, not in**
- Identity Operators   **is, is not**

Arithmetic Operators:

Some basic arithmetic operators are +, -, *, /, %, **, and //. You can apply these operators on numbers as well as variables to perform corresponding operations.

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.	9//2 = 4 and 9.0//2.0 = 4.0

Program:
```
a = 21
b = 10
print "Addition is", a + b
print "Subtraction is ", a - b
print "Multiplication is ", a * b
print "Division is ", a / b
print "Modulus is ", a % b
a = 2
b = 3
print "Power value is ", a ** b
a = 10
b = 4
print "Floor Division is ", a // b
```

Output:

> Addition is 31
> Subtraction is 11
> Multiplication is 210
> Division is 2
> Modulus is 1
> Power value is 8
> Floor Division is 2

Comparison (Relational) Operators

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

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.	(a != b) is true.
< >	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	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:

```
a=20
b=30
if a < b:
    print "b is big"
elif a > b:
    print "a is big"
else:
    print "Both are equal"
```

Output:

> b is big

Assignment Operators

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 / 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:

```
a=82
b=27
a += b
print a
a=25
b=12
a -= b
print a
a=24
b=4
a *= b
print a
a=4
b=6
a **= b
print a
```

Output:

```
109
13
96
4096
```

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 non-zero 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.

Example:

```
a=20
b=10
c=30
if a >= b and a >= c:
    print "a is big"
elif b >= a and b >= c:
    print "b is big"
else:
    print "c is big"
```

Output:

```
c is big
```

Bitwise Operators

Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands.	(a & b) = 12 (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)

Membership Operators

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples.

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:

```
a = 3
list = [1, 2, 3, 4, 5 ];
if ( a in list ):
        print "available"
else:
        print " not available"
```

Output:

available

Identity Operators

Identity operators compare the memory locations of two objects.

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:

```
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"
```

Output:

Line 1 - a and b have same identity
Line 2 - a and b have same identity

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

Operator	Description
()	Parenthesis
**	Exponentiation (raise to the power)
~ x, +x, -x	Complement, unary plus and minus
* / % //	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

Expression:

An expression is a combination of variables constants and operators written according to the syntax of Python language. In Python every expression evaluates to a value i.e., every expression results in some value of a certain type that can be assigned to a variable. Some examples of Python expressions are shown in the table given below.

Algebraic Expression	Python Expression
a x b – c	a * b – c
(m + n) (x + y)	(m + n) * (x + y)
(ab / c)	a * b / c
$3x^2 +2x + 1$	3*x*x+2*x+1
(x / y) + c	x / y + c

Evaluation of Expressions

Expressions are evaluated using an assignment statement of the form

Variable = expression

Variable is any valid C variable name. When the statement is encountered, the expression is evaluated first and then replaces the previous value of the variable on the left hand side. All variables used in the expression must be assigned values before evaluation is attempted.

Example:
a=10
b=22
c=34

```
x=a*b+c
y=a-b*c
z=a+b+c*c-a
print "x=",x
print "y=",y
print "z=",z
```
**Output:**
```
x= 254
y= -738
z= 1178
```

Decision Making:

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.

Decision structures evaluate multiple expressions which produce True or False as outcome. You need to determine which action to take and which statements to execute if outcome is True or False otherwise.

Following is the general form of a typical decision making structure found in most of the programming languages:
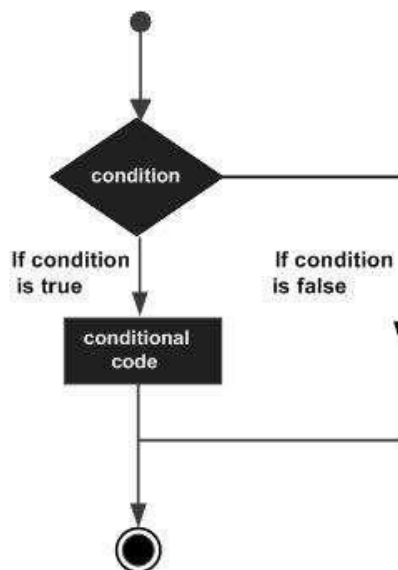

Python programming language assumes any non-zero and non-null values as True, and if it is either zero or null, then it is assumed as False value.

Statement	Description
if statements	**if statement** consists of a boolean expression followed by one or more statements.
if...else statements	**if statement** can be followed by an optional **else statement**, which executes when the boolean expression is FALSE.
nested if statements	You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).

The *if* Statement

It is similar to that of other languages. The **if** statement contains a logical expression using which data is compared and a decision is made based on the result of the comparison.
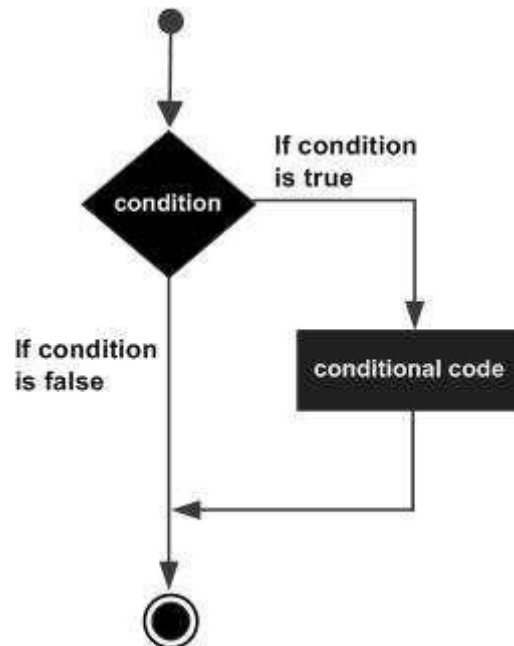
Syntax:

```
if condition:
    statements
```

First, the condition is tested. If the condition is True, then the statements given after colon (:) are executed. We can write one or more statements after colon (:).

Example:

```
a=10
b=15
if a < b:
    print "B is big"
    print "B value is",b
```
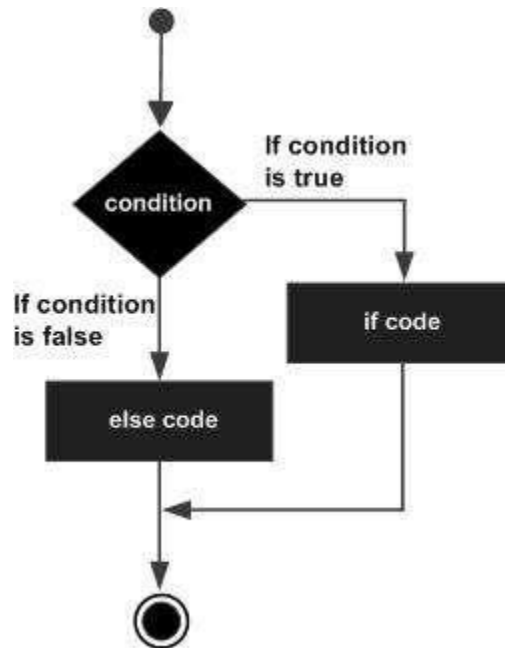
Output:

B is big

B value is 15

The *if ... else* statement

An **else** statement can be combined with an **if** statement. An **else** statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.

The *else* statement is an optional statement and there could be at most only one **else** statement following **if**.

Syntax:

```
if condition:
    statement(s)
else:
    statement(s)
```

Example:

```
a=48
b=34
if a < b:
    print "B is big"
    print "B value is", b
else:
    print "A is big"
    print "A value is", a
print "END"
```

Output:

A is big
A value is 48
END

Q) Write a program for checking whether the given number is even or not.

Program:

```
a=input("Enter a value: ")
if a%2==0:
    print "a is EVEN number"
else:
    print "a is NOT EVEN Number"
```

Output-1:   **Output-2:**

Enter a value: 56   Enter a value: 27
a is EVEN Number   a is NOT EVEN Number

The *elif* Statement

The **elif** statement allows you to check multiple expressions for True and execute a block of code as soon as one of the conditions evaluates to True.

Similar to the **else**, the **elif** statement is optional. However, unlike **else**, for which there can be at most one statement, there can be an arbitrary number of **elif** statements following an **if**.

Syntax:
```
if condition1:
    statement(s)
elif  condition2:
    statement(s)
else:
    statement(s)
```

Example:
```
a=20
b=10
c=30
if a >= b and a >= c:
    print "a is big"
elif b >= a and b >= c:
    print "b is big"
else:
    print "c is big"
```

Output:

c is big

Decision Loops

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement:
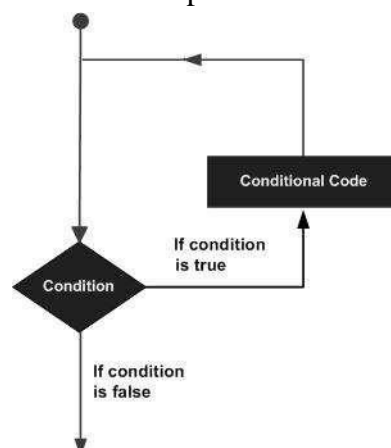
Python programming language provides following types of loops to handle looping requirements.

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loops	You can use one or more loop inside any another while, for loop.

The *while* Loop

A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given condition is True.

Syntax

The syntax of a **while** loop in Python programming language is:

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

Here, **statement(s)** may be a single statement or a block of statements.

The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true. When the condition becomes false, program control passes to the line immediately following the loop.

In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.
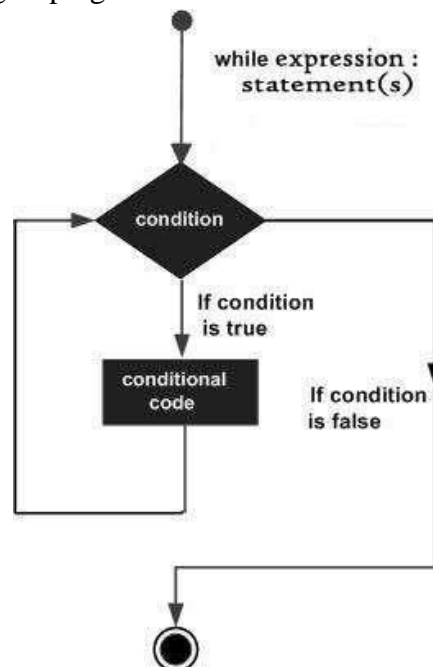
Example-1:

```
i=1
while i < 4:
        print i
        i+=1
        print "END"
```

Example-2:

```
i=1
while i < 4:
        print i
        i+=1
print "END"
```

Output-1:

```
1
END
2
END
3
END
```

Output-2:

```
1
2
3
END
```

Q) Write a program to display factorial of a given number.

   Program:

```
n=input("Enter the number: ")
f=1
while n>0:
    f=f*n
    n=n-1
print "Factorial is",f
```

Output:

Enter the number: 5
Factorial is 120

The *for* loop:

The *for* loop is useful to iterate over the elements of a sequence. It means, the *for* loop can be used to execute a group of statements repeatedly depending upon the number of elements in the sequence. The *for* loop can work with sequence like string, list, tuple, range etc.

The syntax of the for loop is given below:

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

The first element of the sequence is assigned to the variable written after „for‟ and then the statements are executed. Next, the second element of the sequence is assigned to the variable and then the statements are executed second time. In this way, for each element of the sequence, the statements are executed once. So, the *for* loop is executed as many times as there are number of elements in the sequence.

Example-1:

```
for i range(1,5):
        print i
        print "END"
```

Output-1:

```
1
END
2
END
3
END
```

Example-2:

```
for i range(1,5):
        print i
print "END"
```

Output-2:

```
1
2
3
END
```

Example-3:

```
name= "python"
for letter in name:
        print letter
```

Output-3:

```
p
y
t
h
o
n
```

Example-4:

```
for x in range(10,0,-1):
        print x,
```

Output-4:

```
10 9 8 7 6 5 4 3 2 1
```

Q) Write a program to display the factorial of given number.

Program:

```
n=input("Enter the number: ")
f=1
for i in range(1,n+1):
    f=f*i
print "Factorial is",f
```

Output:

Enter the number: 5
Factorial is 120

Nested Loop:

It is possible to write one loop inside another loop. For example, we can write a for loop inside a while loop or a for loop inside another for loop. Such loops are called "nested loops".

Example-1:

```
for i in range(1,6):
    for j in range(1,6):
        print j,
    print ""
```

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

Example-2:

```
for i in range(1,6):
    for j in range(1,6):
        print "*",
    print ""
```

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

Example-3:

```
for i in range(1,6):
    for j in range(1,6):
        if i==1 or j==1 or i==5 or j==5:
            print "*",
        else:
            print " ",
    print ""
```

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

Example-4:

```
for i in range(1,6):
    for j in range(1,6):
        if i==j:
            print "*",
        elif i==1 or j==1 or i==5 or j==5:
            print "*",
        else:
            print " ",
    print ""
```

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

Example-5:

```
for i in range(1,6):
    for j in range(1,6):
        if i==j:
            print "$",
        elif i==1 or j==1 or i==5 or j==5:
            print "*",
        else:
            print " ",
    print ""
```

```
$ * * * *
* $     *
*   $   *
*     $ *
* * * * $
```

Example-6:

```
for i in range(1,6):
    for j in range(1,4):
        if i==1 or j==1 or i==5:
            print "*",
        else:
            print " ",
    print ""
```

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

Example-7:

```
for i in range(1,6):
    for j in range(1,4):
        if i==2 and j==1:
            print "*",
        elif i==4 and j==3:
            print "*",
        elif i==1 or i==3 or i==5:
            print "*",
        else:
            print " ",
    print ""
```

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

Example-8:

```
for i in range(1,6):
    for j in range(1,4):
        if i==1 or j==1 or i==3 or i==5:
            print "*",
        else:
            print " ",
    print ""
```

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

Example-9:

```
for i in range(1,6):
    for c in range(i,6):
        print "",
    for j in range(1,i+1):
        print "*",
    print ""
```

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

Example-10:

```
for i in range(1,6):
    for j in range(1,i+1):
        print j,
    print ""
```

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

Example-11:

```
a=1
for i in range(1,5):
    for j in range(1,i+1):
        print a,
        a=a+1
    print ""
```

```
1
2 3
4 5 6
7 8 9 10
```

1) Write a program for print given number is prime number or not using for loop.

Program:

```
n=input("Enter the n value")
count=0
for i in range(2,n):
    if n%i==0:
        count=count+1
        break
if count==0:
    print "Prime Number"
else:
    print "Not Prime Number"
```

Output:

```
Enter n value: 17
Prime Number
```

2) Write a program print Fibonacci series and sum the even numbers. Fibonacci series is 1,2,3,5,8,13,21,34,55

```
n=input("Enter n value ")
f0=1
f1=2
sum=f1
print f0,f1,
for i in range(1,n-1):
    f2=f0+f1
    print f2,
    f0=f1
    f1=f2
    if f2%2==0:
        sum+=f2
print "\nThe sum of even Fibonacci numbers is", sum
```

Output:

```
Enter n value 10
1 2 3 5 8 13 21 34 55 89
The sum of even fibonacci numbers is 44
```

3) Write a program to print n prime numbers and display the sum of prime numbers.

  Program:

```
n=input("Enter the range: ")
sum=0
for num in range(1,n+1):
    for i in range(2,num):
        if (num % i) == 0:
            break
    else:
        print num,
        sum += num
print "\nSum of prime numbers is",sum
```

  Output:

      Enter the range: 21

      1 2 3 5 7 11 13 17 19

      Sum of prime numbers is 78

4) Using a for loop, write a program that prints out the decimal equivalents of 1/2, 1/3, 1/4, . . . ,1/10

  Program:

```
for i in range(1,11):
    print "Decimal Equivalent of 1/",i,"is",1/float(i)
```

  Output:

      Decimal Equivalent of 1/ 1 is 1.0

      Decimal Equivalent of 1/ 2 is 0.5

      Decimal Equivalent of 1/ 3 is 0.333333333333

      Decimal Equivalent of 1/ 4 is 0.25

      Decimal Equivalent of 1/ 5 is 0.2

      Decimal Equivalent of 1/ 6 is 0.166666666667

      Decimal Equivalent of 1/ 7 is 0.142857142857

      Decimal Equivalent of 1/ 8 is 0.125

      Decimal Equivalent of 1/ 9 is 0.111111111111

      Decimal Equivalent of 1/ 10 is 0.1

5) Write a program that takes input from the user until the user enters -1. After display the sum of numbers.

Program:

```
sum=0
while True:
    n=input("Enter the number: ")
    if n==-1:
        break
    else:
        sum+=n
print "The sum is",sum
```

Output:

Enter the number: 1
Enter the number: 5
Enter the number: 6
Enter the number: 7
Enter the number: 8
Enter the number: 1
Enter the number: 5
Enter the number: -1
The sum is 33

6) Write a program to display the following sequence.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Program:

```
ch='A'
for j in range(1,27):
    print ch,
    ch=chr(ord(ch)+1)
```

7) Write a program to display the following sequence.

A
A B
A B C
A B C D
A B C D E

Program:

```
for i in range(1,6):
    ch='A'
    for j in range(1,i+1):
        print ch,
        ch=chr(ord(ch)+1)
    print ""
```

8) Write a program to display the following sequence.

> **A**
> **B C**
> **D E F**
> **G H I J**
> **K L M N O**

Program:

```
ch='A'
for i in range(1,6):
    for j in range(1,i+1):
        print ch,
        ch=chr(ord(ch)+1)
    print ""
```

9) Write a program that takes input string user and display that string if string contains at least one Uppercase character, one Lowercase character and one digit.

Program:

```
pwd=input("Enter the password:")
u=False
l=False
d=False
for i in range(0,len(pwd)):
    if pwd[i].isupper():
        u=True
    elif pwd[i].islower():
        l=True
    elif pwd[i].isdigit():
        d=True
if u==True and l==True and d==True:
    print pwd.center(20,"*")
else:
    print "Invalid Password"
```

Output-1:

> Enter the password:"Mothi556"
> ******Mothi556******

Output-2:

> Enter the password:"mothilal"
> Invalid Password

10) Write a program to print sum of digits.

 Program:

```
n=input("Enter the number: ")
sum=0
while n>0:
    r=n%10
    sum+=r
    n=n/10
print "sum is",sum
```

 Output:

 Enter the number: 123456789
 sum is 45

11) Write a program to print given number is Armstrong or not.

 Program:

```
n=input("Enter the number: ")
sum=0
t=n
while n>0:
    r=n%10
    sum+=r*r*r
    n=n/10
if sum==t:
    print "ARMSTRONG"
else:
    print "NOT ARMSTRONG"
```

 Output:

 Enter the number: 153
 ARMSTRONG

12) Write a program to take input string from the user and print that string after removing ovals.

 Program:

```
st=input("Enter the string:")
st2=""
for i in st:
    if i not in "aeiouAEIOU":
        st2=st2+i
print st2
```

 Output:

 Enter the string:"Welcome to you"
 Wlcm t y

Arrays:

An array is an object that stores a group of elements of same datatype.

➢ Arrays can store only one type of data. It means, we can store only integer type elements or only float type elements into an array. But we cannot store one integer, one float and one character type element into the same array.

➢ Arrays can increase or decrease their size dynamically. It means, we need not declare the size of the array. When the elements are added, it will increase its size and when the elements are removed, it will automatically decrease its size in memory.

Advantages:

➢ Arrays are similar to lists. The main difference is that arrays can store only one type of elements; whereas, lists can store different types of elements. When dealing with a huge number of elements, arrays use less memory than lists and they offer faster execution than lists.

➢ The size of the array is not fixed in python. Hence, we need not specify how many elements we are going to store into an array in the beginning.

➢ Arrays can grow or shrink in memory dynamically (during runtime).

➢ Arrays are useful to handle a collection of elements like a group of numbers or characters.

➢ Methods that are useful to process the elements of any array are available in „array‟ module.

Creating an array:

Syntax:

arrayname = array(type code, [elements])

The type code „i‟ represents integer type array where we can store integer numbers. If the type code is „f‟ then it represents float type array where we can store numbers with decimal point.

Type code	Description	Minimum size in bytes
„b‟	Signed integer	1
„B‟	Unsigned integer	1
„i‟	Signed integer	2
„I‟	Unsigned integer	2
„l‟	Signed integer	4
„L‟	Unsigned integer	4
„f‟	Floating point	4
„d‟	Double precision floating point	8
„u‟	Unicode character	2

Example:

The type code character should be written in single quotes. After that the elements should be written in inside the square braces [ ] as

a = array („i‟, [4,8,-7,1,2,5,9])

Importing the Array Module:

There are two ways to import the array module into our program.
The first way is to import the entire array module using import statement as,

import array

when we import the array module, we are able to get the „array" class of that module that helps us to create an array.

a = array.array('i', [4,8,-7,1,2,5,9])

Here the first „array" represents the module name and the next „array" represents the class name for which the object is created. We should understand that we are creating our array as an object of array class.

The next way of importing the array module is to write:

from array import *

Observe the „*" symbol that represents „all". The meaning of this statement is this: import all (classes, objects, variables, etc) from the array module into our program. That means significantly importing the „array" class of „array" module. So, there is no need to mention the module name before our array name while creating it. We can create array as:

a = array('i', [4,8,-7,1,2,5,9])

Example:

    from array import *
    arr = array(„i", [4,8,-7,1,2,5,9])
    for i in arr:
            print i,

Output:

    4 8 -7 1 2 5 9

Indexing and slicing of arrays:

An *index* represents the position number of an element in an array. For example, when we creating following integer type array:

a = array('i', [10,20,30,40,50])

Python interpreter allocates 5 blocks of memory, each of 2 bytes size and stores the elements 10, 20, 30, 40 and 50 in these blocks.

10	20	30	40	50
a[0]	a[1]	a[2]	a[3]	a[4]

Example:

    from array import *
    a=array('i', [10,20,30,40,50,60,70])
    print "length is",len(a)
    print " 1st position character", a[1]
    print "Characters from 2 to 4", a[2:5]
    print "Characters from 2 to end", a[2:]
    print "Characters from start to 4", a[:5]
    print "Characters from start to end", a[:]

a[3]=45

a[4]=55

print "Characters from start to end after modifications ",a[:]

Output:

length is 7

1st position character 20

Characters from 2 to 4 array('i', [30, 40, 50])

Characters from 2 to end array('i', [30, 40, 50, 60, 70])

Characters from start to 4 array('i', [10, 20, 30, 40, 50])

Characters from start to end array('i', [10, 20, 30, 40, 50, 60, 70])

Characters from start to end after modifications array('i', [10, 20, 30, 45, 55, 60, 70])

Array Methods:

Method	Description
a.append(x)	Adds an element x at the end of the existing array a.
a.count(x)	Returns the number of occurrences of x in the array a.
a.extend(x)	Appends x at the end of the array a. „x" can be another array or iterable object.
a.fromfile(f,n)	Reads n items from from the file object f and appends at the end of the array a.
a.fromlist(*l*)	Appends items from the *l* to the end of the array. *l* can be any list or iterable object.
a.fromstring(s)	Appends items from string s to end of the array a.
a.index(x)	Returns the position number of the first occurrence of x in the array. Raises „ValueError" if not found.
a.pop(x)	Removes the item x from the array a and returns it.
a.pop( )	Removes last item from the array a
a.remove(x)	Removes the first occurrence of x in the array. Raises „ValueError" if not found.
a.reverse( )	Reverses the order of elements in the array a.
a.tofile(f)	Writes all elements to the file f.
a.tolist( )	Converts array „a" into a list.
a.tostring( )	Converts the array into a string.

1) Write a program to perform stack operations using array.

Program:

```python
import sys
from array import *
a=array('i',[])
while True:
    print "\n1.PUSH 2.POP 3.DISPLAY 4.EXIT"
    ch=input("Enter Your Choice: ")
    if ch==1:
        ele=input("Enter element: ")
        a.append(ele)
        print "Inserted"
    elif ch==2:
        if len(a)==0:
            print "\t STACK IS EMPTY"
        else:
            print "Deleted element is", a.pop( )
    elif ch==3:
        if len(a)==0:
            print "\t STACK IS EMPTY"
        else:
            print "\tThe Elements in Stack is",
            for i in a:
                print i,
    elif ch==4:
        sys.exit()
    else:
        print "\tINVALID CHOICE"
```

Output:

```
1.PUSH 2.POP 3.DISPLAY 4.EXIT
Enter Your Choice: 1
Enter element: 15
Inserted
1.PUSH 2.POP 3.DISPLAY 4.EXIT
Enter Your Choice: 1
Enter element: 18
Inserted
1.PUSH 2.POP 3.DISPLAY 4.EXIT
Enter Your Choice: 3
        The Elements in Stack is 15 18
1.PUSH 2.POP 3.DISPLAY 4.EXIT
Enter Your Choice: 2
Deleted element is 18
```

2) Write a program to perform queue operations using array.

Program:

```python
import sys
from array import *
a=array('i',[])
while True:
    print "\n1.INSERT 2.DELETE 3.DISPLAY 4.EXIT"
    ch=input("Enter Your Choice: ")
    if ch==1:
        ele=input("Enter element: ")
        a.append(ele)
    elif ch==2:
        if len(a)==0:
            print "\t QUEUE IS EMPTY"
        else:
            print "Deleted element is", a[0]
            a.remove(a[0])
    elif ch==3:
        if len(a)==0:
            print "\t QUEUE IS EMPTY"
        else:
            print "\tThe Elements in Queue is",
            for i in a:
                print i,
    elif ch==4:
        sys.exit()
    else:
        print "\tINVALID CHOICE"
```

Output:

```
1.INSERT 2.DELETE 3.DISPLAY 4.EXIT
Enter Your Choice: 1
Enter element: 12
1.INSERT 2.DELETE 3.DISPLAY 4.EXIT
Enter Your Choice: 1
Enter element: 13
1.INSERT 2.DELETE 3.DISPLAY 4.EXIT
Enter Your Choice: 1
Enter element: 14
1.INSERT 2.DELETE 3.DISPLAY 4.EXIT
Enter Your Choice: 3
        The Elements in Queue is 12 13 14
1.INSERT 2.DELETE 3.DISPLAY 4.EXIT
Enter Your Choice: 2
Deleted element is 12
```