Module 6

File Management and Introduction to Python

DATA FILES

Ques 1) Discuss the file management in c. Also discuss the positions in a file.

Or

Discuss the file type, stream and positions of a file.

Or

What is file? Discuss various types of files.

Ans: File and File Management

A file is a collection of bytes stored on a secondary storage device. A file is essentially a serial sequence of bytes. Different types of files store different types of information. **For example**, program files store programs, whereas text files store text. Files are not only used for storing data, programs are also stored in files.

A collection of data or information that has a name, called the **filename**. Almost all information stored in a computer must be in a file. There are many different types of files:

1) Data files,
2) Text files,
3) Program files,
4) Directory files, and so on.

Different types of files store different types of information. **For example,** program files store programs, whereas text files store text. Files are not only used for storing data, programs are also stored in files.

C supports a number of functions that have the ability to perform basic file operations, which include:

1) Opening a file,
2) Reading from a file,
3) Rename a file,
4) Copying a file and etc.,
5) Closing a file.

The standard C language input/output (I/O) functions are a method of sending information to and gaining information from your program as it executes. These functions are built from simple character input and output functions that are easily tailored to meet system hardware requirements.

In order to use files we have to learn about File I/O i.e. how to write information to a file and how to read information from a file. The primary difference between manipulating files and doing terminal I/O is that we must specify in our programs which files we wish to use.

A file is a collection of bytes stored on a secondary storage device. A file is essentially a serial sequence of bytes, as illustrated in **figure 6.1.**

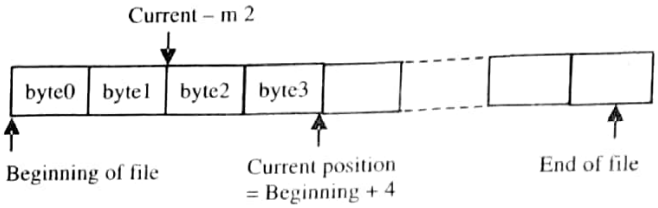


Figure 6.1: Structure of a File

Positions in a File

A file has a beginning and an end, and it has a current position, typically defined as so many bytes from the beginning, as **figure 6.1** illustrates.

The current position is where any file action (a read from the file or a write to the file) will take place.

One can move the current position to any other point in the file. A new current position can be specified as an offset from the beginning of the file or, in some circumstances, as a positive or negative offset from the previous current position.

Streams

The C library provides functions for reading and writing to or from data streams. A stream is an abstract representation of any external source or destination for data, so the keyboard, the command line on your display, and files on disk are all examples of stream.

Therefore one can use the same input/output functions for reading and writing any external device that is mapped to a stream.

The data flow that can be transferred from a program to a file or a vice-versa is called a stream which is a series of bytes.

Types of Files

Essentially there are two kinds of files that programmers deal with text files and binary files. These two classes of files are as follows:

1) **Text Files:** A text file can be a stream of characters that a computer can process sequentially. It is not only processed sequentially but only in forward direction. For this reason a text file is usually opened for only one kind of operation (reading, writing, or appending) at any given time.

   Similarly, since text files only process characters, they can only read or write data one character at a time. (In C Programming Language, Functions are provided that deal with lines of text, but these still essentially process data one character at a time.)

A text stream in C is a special kind of file. Depending on the requirements of the operating system, newline characters may be converted to or from carriage-return/linefeed combinations depending on whether data is being written to, or read from, the file.

Other character conversions may also occur to satisfy the storage requirements of the operating system. These translations occur transparently and they occur because the programmer has signalled the intention to process a text file.

2) **Binary Files:** A binary file is no different to a text file. It is a collection of bytes. In C programming language a byte and a character are equivalent. Hence a binary file is also referred to as a character stream, but there are two essential differences.

No special processing of the data occurs and each byte of data is transferred to or from the disk unprocessed.

C programming language places no constructs on the file, and it may be read from, or written to, in any manner chosen by the programmer.

Binary files can be either processed sequentially or, depending on the needs of the application, they can be processed using random access techniques. In C Programming Language, processing a file using random access techniques involves moving the current file position to an appropriate place in the file before reading or writing data. This indicates a second characteristic of binary files. They are generally processed using read and write operations simultaneously.

For example, a database file will be created and processed as a binary file. A record update operation will involve locating the appropriate record, reading the record into memory, modifying it in some way, and finally writing the record back to disk at its appropriate location in the file. These kinds of operations are common to many binary files, but are rarely found in applications that process text files.

Ques 2) Explain different operations performed on File.

Ans: Operations Performed on File

1) **Opening a File:** To open a file one need to use the fopen(). Once one has opened a file, he/she can use the FILE pointer to let the compiler perform input and output functions on the file.

Syntax:
FILE *fp;
fp = fopen("filename", "mode");

2) **Reading from a File:** fread() function reads the file contents.

Syntax:
fread(void *ptr, size_t size, size_t nmemb, FILE *stream)

Where,

i) **ptr:** The array where the elements will be stored.
ii) **size:** The size of each element in bytes.
iii) **nmemb:** The number of elements to read.
iv) **stream:** The stream to read.

The fread function returns the number of elements read. The fread function will return zero if nmemb is zero or size is zero.

3) **Rename a File:** If anyone wants to change the name of a file, rename() function can be used. It renames a file.

Syntax:
rename(const char *oldname, const char *newname);

4) **Closing a File:** When a program has finished with reading/writing to a file, it must be closed so that the communication area (buffer) will be free and available for other file. The C library function fclose() closes the stream.

Syntax:
fclose(File *fp);

Ques 3) Explain in detail various functions used for sequential file manipulations.

Ans: Functions for Sequential File Manipulation

The various functions used in sequential file manipulation are given in the **table 6.1**:

Table 6.1: File Functions

Function	Operation
fopen()	Creates a new file for read/write operation.
fclose()	Closes a file associated with file pointer.
closeall()	Closes all opened files with fopen().
fgetc()	Reads the character from current pointer position and advances the pointer to next character.
getc()	Same as fgetc().
fprintf()	Writes all types of data values to the file.
fscanf()	Reads all types of data values from a file.
putc()	Writes character one by one to a file.
fputc()	Same as putc().
gets()	Reads string from the file.
puts()	Writes string to the file.
putw()	Writes an integer to the file.
getw()	Reads an integer from the file.
fread()	Reads structured data written by fwrite() function.
fwrite()	Writes block of structured data to the file.
fseek()	Sets the pointer position anywhere in the file.
feof()	Detects the end of file.
ferror()	Reports error occurred while read/write operations.
perror()	Prints compilers error messages alongwith user defined messages.
ftell()	Returns the current pointer position.
rewind()	Sets the record pointer at the beginning of the file.
unlink()	Removes the specified file from the disk.
rename()	Changes the name of the file.

Ques 4) Explain various file opening modes.

Ans: File Opening Modes

When anyone opens a file, he/she must specify how it is to be opened. This means whether to create it from new or overwrite and whether it's text or binary, read or write and if user wants to append to it. This is done using one or more file mode specifiers which are single letters "r", "b", "w", "a" and + (in combination with the other letters). Some common file opening modes are shown in **table 6.2**:

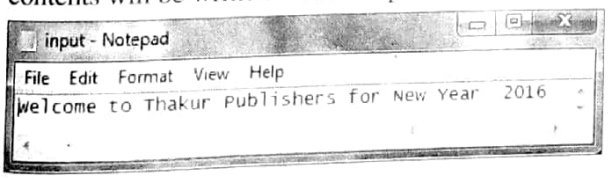
Table 6.2: File Opening Modes

Mode	Meaning
Text File Modes	
"r"	Open file for reading and file must exist;
"w"	Open file for writing. If file does not exist it is created or if life already exists its content is erased.
"a"	Open file for appending. It adds all information at the end of the file leaving old data untouched. If file does not exist it is created.
"r+"	Open file for reading and writing and file must exist.
"w+"	Open file for writing and reading. If file does not exist it is created or if file already exist its content is erased.
"a+"	open for reading and writing (append if file exists)
Binary File Modes	
"rb"	Open binary file for reading
"wb"	Create binary file for writing
"ab"	Append to a binary file
"rb+' or "r+b"	Open binary file for read/write
"wb+" or "w+b"	Create binary file for read/write
"ab+" or "a+b"	Open binary file for read/write

Program: Illustrating Use of w+

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  FILE * fp;
  fp = fopen ("input.txt", "w+");
  fprintf(fp, "%s %s %s %d", "Welcome", "to", "Thakur Publishers for New Year
  ", 2016);
  fclose(fp);
  return(0);
}
```

Output: After compiling and running the above program, the contents will be written in a file input.txt as shown below:



Ques 5) Write the short notes on File I/O and console I/O (May 2011 [5])

Ans: File I/O

A file represents a sequence of bytes, does not matter if it is a text file or binary file. C programming language provides access on high level functions as well as low level (OS level) calls to handle file on storage devices.

The basic steps for using a File in C are as below:

1) Create a variable of type "FILE*".
2) Open the file using the "fopen" function and assign the "file" to the variable.
3) Check to make sure the file was successfully opened by checking to see if the variable == NULL. If it does, an error has occurred.
4) Use the fprintf or fscanf functions to write/read from the file. Usually these function calls are placed in a loop. In the case of reading data, usually, the data is read in and placed in an array.

Console I/O

The screen and keyboard together are called a console. Console I/O in general means communications with the computer's keyboard and display. Console I/O functions can be further classified into two categories:

1) Formatted Console I/O functions
2) Unformatted Console I/O functions

The functions available under each of these two categories are shown in **table 6.3**:

Table 6.3: Console Input/ Output Functions

Formatted Functions			Unformatted Functions		
Type	**Input**	**Output**	**Type**	**Input**	**Output**
char	scanf()	printf()	char	getch() getche() getchar()	putch() putchar()
int	scanf()	printf()	int	–	–
float	scanf()	printf()	float	–	–
string	scanf()	printf()	string	gets()	puts()

The basic difference between them is that the formatted functions allow the input read from the keyboard or the output displayed on the VDU (Visual Display Unit) to be formatted as per our requirements.

For example, if values of average marks and percentage marks are to be displayed on the screen, then the details like where this output would appear on the screen, how many spaces would be present between the two values, the number of places after the decimal points, etc. can be controlled using formatted functions.

Ques 6) Explain the operation of following I/O functions: (May 2011 [5], Dec 2012 [10])

i) fopen() and fclose()

Ans: fopen()

To open a file one need to use fopen() function, which returns a FILE pointer. Once user has opened a file, he/she can use the FILE pointer to let the compiler perform input and output functions on the file.

Syntax:

```
FILE *fp;
fp=fopen("filename", "mode");
```

Where,
1) The first statement declares the variable **fp** as a file pointer to the data type FILE.
2) The second statement opens the file named filename and assigns an identifier to the FILE type pointer fp. The second statement also specifies the purpose of opening the file. The mode does this job.

For example, let consider the following code:

```
FILE *fp;
fp = fopen("c:\\test.txt", "r");
```

This code will open test.txt for reading in text mode. To open a file in a binary mode one must add a **b** to the end of the mode string; for example, "rb" (for the reading and writing modes, one can add the b either after the plus sign- "r + b" – or before – "rb +" for binary files).

fclose()

When a program has finished with reading/writing to a file, it must be closed so that the communication area (buffer) will be free and available for other file. The input output library function fclose() is used to close a file.

Syntax:

```
fclose( File *fp);
```

The fclose() takes a pointer to a FILE as a parameter and attempts to close the specified file. If the file is close successfully, fclose() frees up the memory allocated to the FILE struct and returns a value of 0.

ii) getc()

Ans: getc()

This library function gets the next character from the specified stream and advances the position indicator for the stream. The getc() returns a character from the specified FILE. The getc function is equivalent to fgetc, except that it may be implemented as a macro.

If it is implemented as a macro, the stream argument may be evaluated more than once, so the argument should never be an expression with side effects (i.e. have an assignment, increment, or decrement operators, or be a function call).

Syntax:

```
int getc(FILE *stream);
```

iii) putw()

Ans: putw()

The putw() function is used to write integers to the file.

Syntax:

```
putw(int number, FILE *fp);
```

The putw() function takes two arguments:
1) **First** is an integer value to be written to the file and
2) **Second** is the file pointer where the number will be written.

iv) fscanf()

Ans: fscanf()

The fscanf function works exactly like the scanf function except that it reads the information from the stream specified by stream instead of standard input device.

Syntax:

```
fscanf (FILE *stream, const char *format,.....);
```

Where,
1) **stream** is the pointer to a FILE object that identifies the stream.
2) **format** is the C string that contains one or more of the following items – Whitespace character, Non-whitespace character and Format specifiers. A format specifier will be as **[=%[*][width][modifiers]type=]**, which is shown in table below:

Argument	Description
*	This is an optional starting asterisk indicates that the data is to be read from the stream but ignored, i.e. it is not stored in the corresponding argument.
width	This specifies the maximum number of characters to be read in the current reading operation.
modifiers	Specifies a size different from int (in the case of d, i and n), unsigned int (in the case of o, u and x) or float (in the case of e, f and g) for the data pointed by the corresponding additional argument: h : short int (for d, i and n), or unsigned short int (for o, u and x) l : long int (for d, i and n), or unsigned long int (for o, u and x), or double (for e, f and g) L : long double (for e, f and g)
type	A character specifying the type of data to be read and how it is expected to be read. See next table.

v) ftell()

Ans: ftell()

Gets the current position of a file pointer.

Syntax:

```
long ftell( FILE *stream );
```

ftell returns the current position of the file being pointed by the file pointer.

For example, in the following statements:

```
FILE *fp = fopen("input.txt", "w");
pos= ftell(fp);
```

Whenever a file is opened in read or writes mode the file poiner is placed at the beginning of the file i.e., at the first character which is at position 0 so pos will have 0 which ftell returns.

Ques 7) Explain the following with syntax: **(Dec 2015 [10])**

i) fprintf

Ans: fprintf
This library function sends formatted output to a stream.

Syntax:
int fprintf(FILE *stream, const char *format, ...)

where,
1) **stream:** This is the pointer to a FILE object that identifies the stream.
2) **format:** This is the C string that contains the text to be written to the stream. It can optionally contain embedded format tags that are replaced by the values specified in subsequent additional arguments and formatted as requested.

ii) fwrite.

Ans: fwrite
This library function writes data from the array pointed to, by ptr to the given stream. The functions fwrite are used for writing data to the file opened by fopen function.

Syntax:
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)

Where,
1) **ptr:** This is the pointer to the array of elements to be written.
2) **size:** This is the size in bytes of each element to be written.
3) **nmemb:** This is the number of elements, each one with a size of size bytes.
4) **stream:** This is the pointer to a FILE object that specifies an output stream.

Ques 8) Discuss the random access to files.

Ans: Random Access to a File
In order to read a data item from anywhere (or randomly) in a file, we have to move the file pointer to the beginning of that data item. To accomplish this task, one can use fseek() function.

fseek() Function
This function sets the file position indicator for the stream pointed to by stream or it seeks a specified place within a file and modify it.

Syntax:
int fseek(file *stream, long offset, int wherefrom);

The fseek() function takes three arguments, where
1) First argument stream is a file pointer,
2) Second argument offset is variable of type long integer that specifies the number of bytes by which file pointer is to be moved,
3) Third argument "wherefrom" specifies from which position the offset is measured. Its value can be 0, 1 or 2, representing three symbolic constants SEEK_SET, SEEK_CUR and SEEK_END respectively.

The various values for argument **wherefrom** are listed in **table 6.4**:

Table 6.4: Various Values of wherefrom for fseek()Function

Constant	Offset is Measured From
SEEK_SET	Beginning of the file.
SEEK_CUR	Current position of the file.
SEEK_END	End of the file.

For example, consider a file stream variable stream1 has been declared of type FILE and has been opened for reading. If we are interested to read the 21st byte of the file, the fseek can be used as:

```
fseek(stream1, 20, SEEK_SET);
```

Alternatively, it can also be written as

```
fseek(stream1, 20, 0);
```

It will forward the file pointer by 20 bytes with respect to the beginning of the file, meaning that 21st byte of stream1 will be read whenever any reading operation is applied on the file.

In order to read the last character of the file, the fseek can be used as:

```
fseek(stream1, -1, 2);
```

This will shift the file pointer one position before the end of the file and hence last character can be read now.

Example: Illustrating fseek() function.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    FILE *f;
    clrscr();
    f=fopen("hello.txt", "w");
    fputs("Hello Sanjay", f);
    fseek(f,6,SEEK_SET);
    fputs("India", f);
    fclose(f);
    getch();
    return 0;
}
```

Explanation: In the following code, the function **fopen (file, "w")** open a file and allow to perform write operations into the file. The **fputs()** function writes the string "Hello Sanjay" into the file. Then we have used **fseek(f , 6 , SEEK_SET)** function which sets the file position indicator to 6 and the fputs() function write the string "Indiay" in place of "Sanjay". Here string saved in the text file is Indiay and y in india is not replaced because word india has 5 character and word sanjay has six character.

Ques 9) What is FILE pointer in c programming language?

Ans: FILE Pointer

FILE pointer is struct data type which has been defined in standard library stdio.h. This data type points to a stream or a null value. It has been defined in stdio.h as

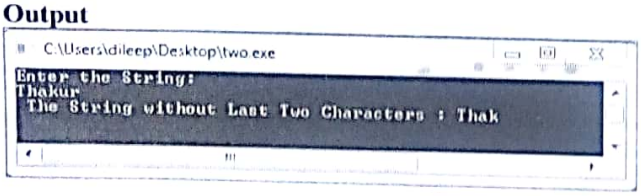
```
typedef struct{
   short        level;
   unsigned     flags;
   char         fd;
   unsigned char  hold;
   short        bsize;
   unsigned char  *buffer, *curp;
   unsigned     istemp;
   short        token;
} FILE;
```

Ques 10) Without using string.h, write a program which reads a string & then removes the last two characters of the string and displays that. The length of input string cannot be asked from the user. For example, if inputted string was "ABCDE", output should be "ABC". (Dec 2010 [10])

Ans: Program to Print String after Removing Last Two Characters

```
#include<stdio.h>
char* LastcharDel(char* name);
int main () {
   char name[20];
   printf("Enter the String:\n");
   scanf("%s", name);
   LastcharDel(name);
   printf(" The String without Last Two Characters :
      %s\n\n", name);
   return 0; }
char* LastcharDel(char* name) {
   int i = 0;
   while(name[i] != '\0')    {
      i++;          }
   name[i-2] = '\0';
   return name; }
```

Output



Ques 11) Write a function in C language that takes two matrices A (M × N) and B (M × N) of integers as an argument and returns the two matrices C (M × N) and D (M × N) where, C = A + B and D = A * B.

Ans: Functions for Addition and Multiplication of Two Matrices

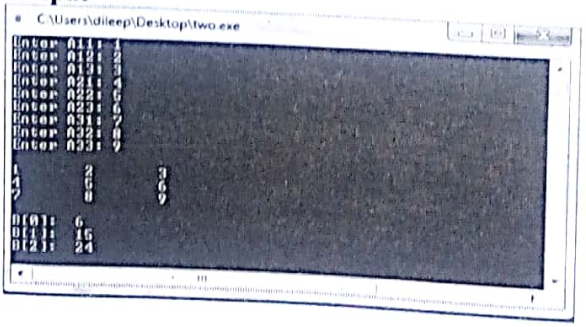
```
int **add_matrix(int M, int N, int **A, int M, int N, int **B)
{
      int c=0,d=0;
      int **C = (int **) malloc(sizeof(int *)*M);
      for( c = 0 ; c < M; c++ )
      {
         for( d = 0 ; d < N ; d++ )   {
            C[c][d] = *(*(A+c)+d)+*(*(B+c)+d);     }}
      return C;}

int **mul_matrix(int M, int N, int **A, int M, int N, int **B)
{  int c=0,d=0;
      int **D = (int **) malloc(sizeof(int *)*M);
      for( c = 0 ; c < M; c++ ) {
            for( d = 0 ; d < N; d++ ) {
               D[c][d] = (*(*(A+c)+d))*(*(*(B+c)+d)); }}
      return D;}
```

Ques 12) Create a two dimensional array A, of size 3 × 3 of integer numbers using pointers in C language. Then create another one dimensional array B, of size 3 that holds the row-wise summation of all the row elements of array A, using pointers.

Ans: Program Displays Summation of All Row Elements of Array A

```
#include <stdio.h>
#include <conio.h>
int main() {
      int a, i, j, A[3][3], B[3], sum;
      for (i = 0; i < 3; i++) {
            for (j = 0; j < 3; j++) {
                  printf("Enter A%d%d: ",i+1,j+1);
                  scanf("%d",&a);
                  *(*(A + i) + j)=a;   } }
      printf("\n");
      for (i = 0; i < 3; i++) {
            for (j = 0; j < 3; j++) {
                  printf("%d\t", *(*(A + i) + j)); }
            printf("\n"); }
      printf("\n");
      for (i = 0; i < 3; i++)
      {
            sum=0;
            for (j = 0; j <3; j++)
            {
                  sum+=*(*(A + i) + j);}
            B[i]=sum;
      }
      for (i = 0; i <3; i++)
      {
            printf("B[%d]:  %d\n", i, B[i]);}
      return 0;}
```

Output

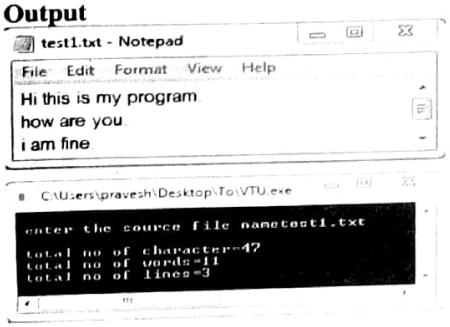


Ques 13) Write a 'C' Program to read a file and print number of characters, words and lines in it.

Ans: C Program to Read a File and Printing Number of Characters, Words and Lines

```
#include<stdio.h>
#include<conio.h>
main() {
    int noc=0,now=0,nol=0;
    FILE *fw,*fr;
    char fname[20],ch;
    printf("\n enter the source file name");
    gets(fname);
    fr=fopen(fname,"r");
    if(fr ==NULL) {
        printf("\n error \n");
        exit(0); }
    ch=fgetc(fr);
    while(ch!=EOF) {
        noc++;
        if(ch==' ') {
            now++; }
        if(ch=='\n') {
            nol++;
            now++; }
        ch=fgetc(fr);}
    fclose(fr);
    printf("\n total no of character=%d",noc);
    printf("\n total no of words=%d",now);
    printf("\n total no of lines=%d",nol);
    getch(); }
```

Output

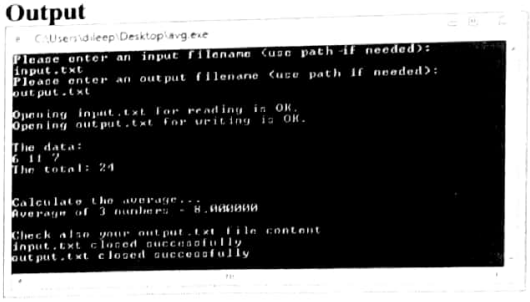


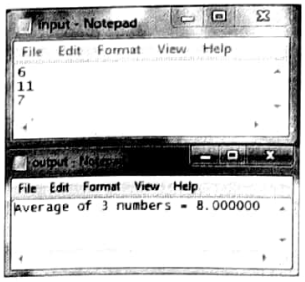
Ques 14) Write a C program which reads all numbers from a given input file and stores average of those numbers in an output file. Both file names are input from user. **(Dec 2010 [10])**

Ans: Program to Print Average of Input Numbers

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int value, total = 0, count = 0;
    FILE * fileptrIn, * fileptrOut;
    errno_t err = 0, err1 = 0;
    char filenameIn[50], filenameOut[50];
    printf("Please enter an input filename (use path if
    needed):\n");
    scanf_s("%s", filenameIn, 50);
    printf("Please enter an output filename (use path if
    needed):\n");
    scanf_s("%s", filenameOut, 50);
    err = fopen_s(&fileptrIn, filenameIn,"r");
    err1 = fopen_s(&fileptrOut, filenameOut,"w");
    if(err !=0 )
    {
        printf("\nError opening %s for reading.\n",
        filenameIn);
        exit (1); }
    else
        printf("\nOpening %s for reading is OK.\n",
        filenameIn);

    if(err1 != 0)
    {
        printf("Error opening %s for writing.\n",
        filenameOut);
        exit (1);}
    else
        printf("Opening %s for writing is OK.\n",
        filenameOut);
        printf("\nThe data:\n");
    while(EOF != fscanf_s(fileptrIn, "%i", &value))
    {
        printf("%d ", value);
        total += value;
        ++count;}
    printf("\nThe total: %d\n", total);
    printf("\n\nCalculate the average...\n");
    fprintf(fileptrOut, "Average of %i numbers = %f \n",
    count, total/(double)count);
    printf("Average of %i numbers = %f \n\n", count,
    total/(double)count);
    printf("Check also your %s file content\n",
    filenameOut);
    if(fclose(fileptrIn) == 0)
    printf("%s closed successfully\n", filenameIn);
    if(fclose(fileptrOut) == 0)
    printf("%s closed successfully\n", filenameOut);
    return 0;
}
```

Output



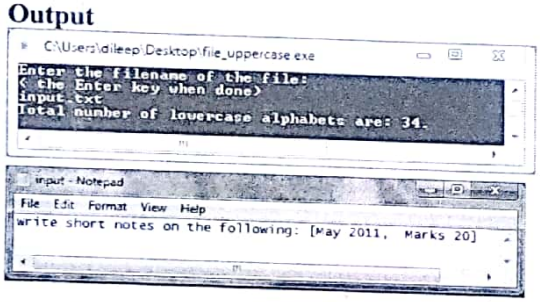


Ques 15) Write a C program which counts the total number of lowercase alphabets in a given input file. Name of the input file should be taken from user.
(Dec 2011 [10])

Ans: Program to Count Total Number of Lowercase Alphabets

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
int main()
{
  char Character = 0;
  int lower_case = 0;
  char user_filename[100];
  char user_filecontent[100];
  printf("Enter the filename of the file:\n");
  printf("( the Enter key when done)\n");
  gets(user_filename);

  FILE *fp;
  fp = fopen (user_filename, "r");

  if (fp == NULL)
  {
    printf("\nError, Unable to open the file for reading\n");
  }

  while((Character = fgetc(fp)) != EOF)
  {

    if (islower(Character))
    {
      lower_case++;
    }  }
  fclose(fp);
  printf("Total number of lowercase alphabets are: %d.\n",
lower_case);
  return 0;}
```

Output

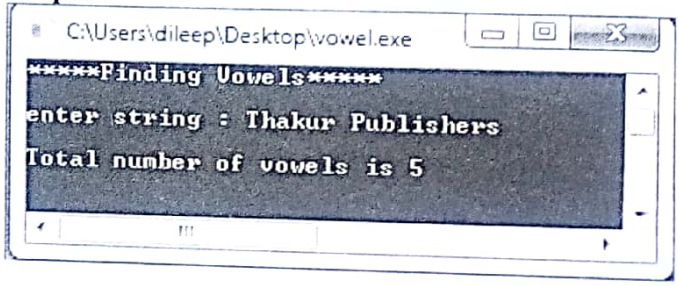


Ques 16) Write a program to find and print number of vowels in a string entered by user.
(Dec 2013 [10])

Ans: Program to Find and Print Number of Vowels

```
#include <stdio.h>
#include <conio.h>
void main()
{
   int chasc, i, cnt=0;
   char ch, str[40]="";
   printf("*****Finding Vowels*****\n\n");
   printf("enter string : ");
   scanf("%s",str);.
   for(i=0;str[i]!='\0';i++)
   {
        chasc = str[i];
        switch(chasc)
        {
            case'a' :
            case'A' :
            case'e' :
            case'E' :
            case'i' :
            case'I' :
            case'o' :
            case'O' :
            case'u' :
            case'U' :
            cnt++;
       }}
printf("\nTotal number of vowels is %d",cnt);
getch();
}
```

Output



PYTHON

Ques 17) What is Python? Name some of the features of Python.

Or

Write a short on python. List the characteristics of python language.

Ans: Python

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Features/Characteristics of Python

1) **Easy-to-Learn:** Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.

2) **Easy-to-Read:** Python code is more clearly defined and visible to the eyes.

3) **Easy-to-Maintain:** Python's source code is fairly easy-to-maintain.
4) **A Broad Standard Library:** Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
5) **Interactive Mode:** Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
6) **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
7) **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
8) **Databases:** Python provides interfaces to all major commercial databases.
9) **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of UNIX.
10) **Scalable:** Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below:

1) It supports functional and structured programming methods as well as OOP.
2) It can be used as a scripting language or can be compiled to byte-code for building large applications.
3) It provides very high-level dynamic data types and supports dynamic type checking.
4) It supports automatic garbage collection.
5) It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Ques 18) How many different ways to start Python?

Or

Write the step to start the python. Also explain the program structure and execution of the python.

Ans: Different Ways to Start Python

There are three different ways to start Python:

1) **Interactive Interpreter:** One can start Python from UNIX, DOS, or any other system that provides you a command-line interpreter or shell window.

   Enter python the command line. Start coding right away in the interactive interpreter.

```
$python          # Unix/Linux
                  Or
python%          # Unix/Linux
                  Or
C:>python        # Windows/DOS
```

2) **Script from the Command-Line:** A Python script can be executed at command line by invoking the interpreter on your application, as in the following:

```
$python script.py        # Unix/Linux
                          Or
python% script.py        # Unix/Linux
                          Or
C:>python script.py      # Windows/DOS
```

3) **Integrated Development Environment:** One can run Python from a Graphical User Interface (GUI) environment as well, if you have a GUI application on your system that supports Python.

   Following are the various versions of IDE for python:

   i) **UNIX:** IDLE is the very first Unix IDE for Python.
   ii) **Windows:** PythonWin is the first Windows interface for Python and is an IDE with a GUI.
   iii) **Macintosh:** The Macintosh version of Python along with the IDLE IDE is available from the main website, downloadable as either MacBinary or BinHex'd files.

Python Program Structure

A Python program is a sequence of statements. It executes this sequence of statements in a specific, consistent, and predictable order and all the statement contains zero or more expressions.

A Python expression describes a computation, or operation, performed on data. An expression is some text a programmer writes, and a value is Python's internal representation of a piece of data. Evaluating an expression computes a Python value.

Execution of Python Program

Python executes a program by executing the program's statements one by one until there are no more statements left to execute. In general, Python executes statements from top to bottom.

Python executes a statement by evaluating its expressions to values one by one, then performing some operation on those values.

Python evaluates an expression by first evaluating its sub-expressions, then performing an operation on the values.

Ques 19) What do you mean by variables in python? How values will be assigning to these variables?

Or

What are the variables and identifiers in python? Write the rules in python for the identifiers.

Or

What is assignment operator, literals in the python? Explain with example.

Ans: Variables

Variables are named locations which are used to store references to the object stored in memory. The names we choose for variables and functions are commonly known as 'Identifiers'.

Rules in Python for the Identifiers

In python Identifiers must obey the following **rules**:

1) All identifiers must start with letter or underscore (_) , one can't use digits. **For example,** my_var is valid identifier and 1 is not valid identifier
2) Identifiers can contain letters, digits and underscores (_).

3) They can be of any length.
4) Identifier cannot be a keyword (keywords are reserved words that Python uses for special purpose).

In python one do not need to declare types of variable ahead of time. Interpreter automatically detects the type of the variable by the data it contains.

Values/Literals
Values are basic things that programs works with. **For example,** 1, 11, 3.14, "hello" are all values. In programming terminology they are also commonly known as **literals**.

Literals can be of different type, **for example,** 1, 11 are of type int , 3.14 is float and "hello" is string .

Assigning Values into Variables/Assignment Operator
To assign value to a variable equal sign (=) is used. '=' is also known as **assignment operator**. Following are some **examples** of variable declaration:

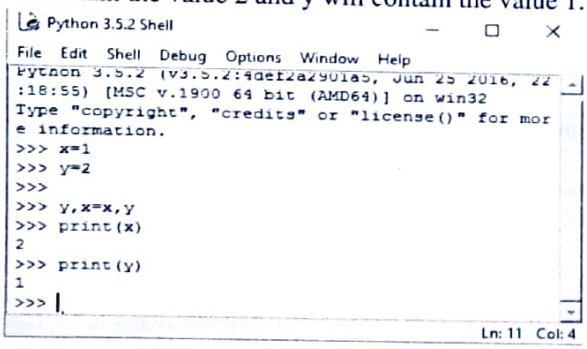
```
x = 100                          # x is integer
pi = 3.14                        # pi is float
em_pname = "python is great"     # empname is string
a = b = c = 100 # this statement assign 100 to c, b and a.
```

Python allow **simultaneous assignment**. This statement tells the python to evaluate all the expression on the right and assign them to the corresponding variables on the left. Simultaneous assignment is helpful to swap values of two variables.

Syntax:

var1, var2, ..., varn = exp1, exp2, ..., expn

Consider the following **example** of simultaneously assignment of x and y, where x contains the value 1 and y contains the value 2. After simultaneously assignment x will contain the value 2 and y will contain the value 1.



Ques 20) What are keywords? Write some keywords used in python.

Ans: Keywords
Keywords define the language's rules and structure, and they cannot be used as variable names. Python has 'twenty-nine' keywords:

and	def	exec	if	not	return
assert	del	finally	import	or	try
break	elif	for	in	pass	while
class	else	from	is	print	yield
continue	except	global	lambda	raise	

Ques 21) What are expressions and statements in python? How expressions can be evaluated?

Or

Explain expression used in python. Also explain the Boolean expression with example.

Or

Define expression. How compound expressions are evaluated in python?

Ans: Expressions
An expression is a combination of values, variables, and operators. If one types an expression on the command line, the interpreter evaluates it and displays the result.

Consider the following code **example** the expression '1 + 1' is evaluated as 2:

```
>>> 1 + 1
2
```

Although expressions contain values, variables, and operators, not every expression contains all of these elements. A value all by itself is considered an expression, and so is a variable. Following is an **example** of it:

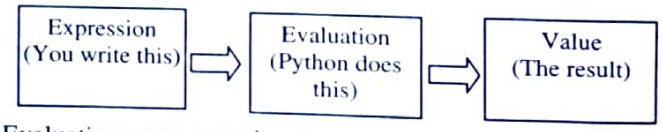
```
>>> 17
17
```

Boolean Expression
A Boolean expression is an expression that is either true or false. One way to write a Boolean expression is to use the operator ==, which compares two values and produces a Boolean value.

For example, in the following first statement, the two operands are equal, so the value of the expression is True; in the second statement, 5 is not equal to 6, so we get False.

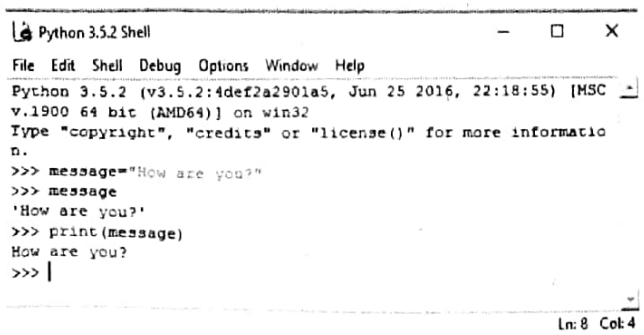
```
>>> 5 == 5
True
>>> 5 == 6
False
```

Evaluation of Expressions
The process that the computer uses to turn expression into whatever that expression means is called 'evaluation'. When the process of evaluation is complete, you are left with a single "value".



Evaluating an expression is not quite the same thing as printing a value. Consider the following **example**:

```
>>> message = "How are you?"
>>> message
'How are you?'
>>> print (message)
How are you?
>>>
```



When the Python interpreter displays the value of an expression, it uses the same format one would use to enter a value. In the case of strings, that means that it includes the quotation marks. But if one uses a print statement, Python displays the contents of the string without the quotation marks as shown above.

Evaluation of Compound Expression

To evaluate a compound expression to a value,

1) Use order of operations to identify the main operator (the last operator that you'll apply).
2) Identify the operands to the main operator. Then evaluate this expression (the main operator and its two operands) as you would evaluate a binary expression.

Below are **examples** of evaluating compound expressions:

```
3 * 6 + 7
3 * 6 + 7
3 * 6 + 7
18 + 7
18 + 7
25
```

This example contains some extra underlining to emphasise the following. To evaluate 3 * 6 + 7, it is necessary to first evaluate its left-hand side, 3 * 6. To evaluate 3 * 6, it is necessary to first evaluate its left-hand side, 3.

Statements

A statement is an instruction that the Python interpreter can execute.

When one types a statement on the command line, Python executes it and displays the result, if there is one. The two frequently used statements in python are print statement and assignment statement. The result of a print statement is a value. Assignment statements do not produce a result.

A **script** usually contains a sequence of statements. If there is more than one statement, the results appear one at a time as the statements execute.

For example, consider the following code segment with print and assignment statements:

```
print(1)
x = 2
print(x)
```

It will produce the output as follows:

```
1
2
```

Ques 22) List and explain the standard data types used in python with the use of examples.

Or

What are data types? Explain with example.

Ans: 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:

1) **Numbers:** Number data types store numeric values. Number objects are created when you assign a value to them. Python supports three different numerical types:
   i) **int (signed integers)** for integer values such as 45.
   ii) **float (floating point real values)** for floating point values such as 2.3 .
   iii) **complex (complex numbers)** for complex numbers such as 3+2j .
2) **String:** 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. Python do not have any separate data type for characters so they are represented as a single character string.

   Syntax:

   String_var= "Val"

   The **example** below illustrating the string variable and its value:

   ```
   name = "tom"      # a string
   mychar = 'a'      # a character
   ```
3) **List:** List type is another sequence type defined by the list class of python. List allows one to add, delete or process elements in very simple ways. List is very similar to arrays. To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type. One can create list using the following syntax.

   Syntax:

   List_var= [Val1, Val2, Val3, Val4.....]

   Here each elements in the list is separated by comma and enclosed by a pair of square brackets ([]). Elements in the list can be of same type or different type. **For example,** the list l = ["this is a string", 12] contains the two different types of values, one is string type and other is integer type.
4) **Tuple:** A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

   Syntax:

   tuple = (val1, val2,...)

The main differences between lists and tuples is that lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as read-only lists. **For example,** following is the tuple created with different types of values:

tuple = ('abcd', 786 , 2.23, 'john', 70.2)

5) **Dictionary:** Dictionary is a python data type that is used to store key value pairs. It enables you to quickly retrieve, add, remove, modify, values using key. Dictionary is very similar to what we call associative array or hash on other languages. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries can be created using pair of curly braces ({}). Each item in the dictionary consist of key, followed by a colon, which is followed by value. And each item is separated using commas (,).

Syntax:

Dict_var={key1:val1, key2:val2,....}

For example, the following dictionary named as **friends** is a dictionary with two items. One point to note that key must be of hash table type, but value can be of any type. Each key in the dictionary must be unique.

```
friends = {
'tom' : '111-222-333',
'jerry' : '666-33-111'
}
```

Ques 23) What is operator? List the various types of operator used in python.

Or

Explain the following operators in python

i) Logical Operators

ii) Bitwise Operators

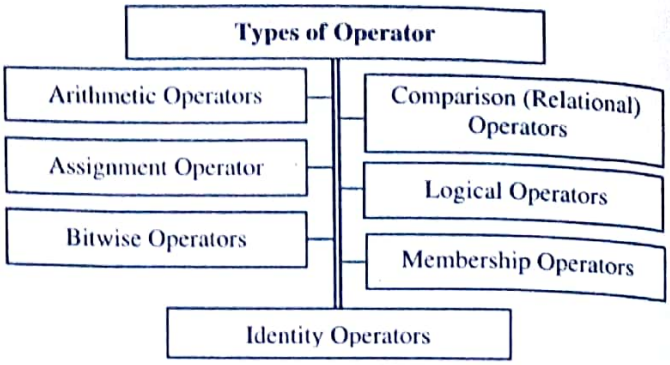
iii) Membership Operators

Ans: Operator

Operators are special symbols that represent computations like addition and multiplication. The values the operator uses are called operands. **For example,** consider the expression 4 + 5 = 9. Here, 4 and 5 are called 'operands' and + is called 'operator'. Python has the different types of operators which allow one to carry out required calculations in the program.

Types of Operator

Python language supports the following types of operators.

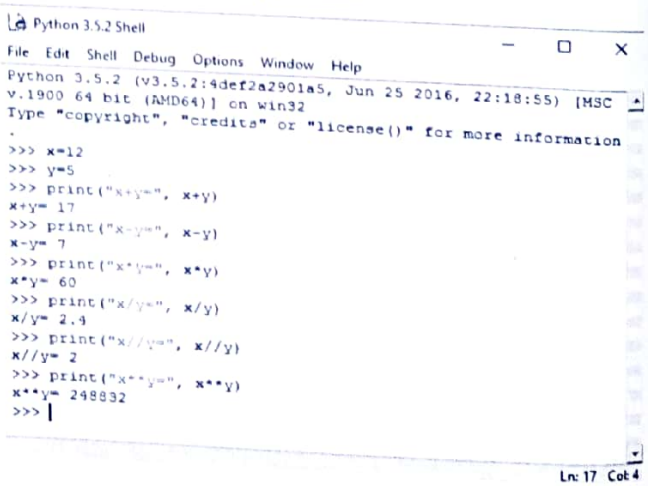


1) **Arithmetic Operators:** Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc. The following **table 6.5** illustrating the various types of arithmetic operators used in python:

Table 6.5: Arithmetic Operators in Python

Operator	Meaning	For Example
+	Add two operands or unary plus	x + y; +2
–	Subtract right operand from the left or unary minus	x – y; –2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x/y
%	Modulus – remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division – division that results into whole number adjusted to the left in the number line	x//y
**	Exponent – left operand raised to the power of right	x ** y (x to the power y)

For example, the following code segment illustrates the use of arithmetic operators:

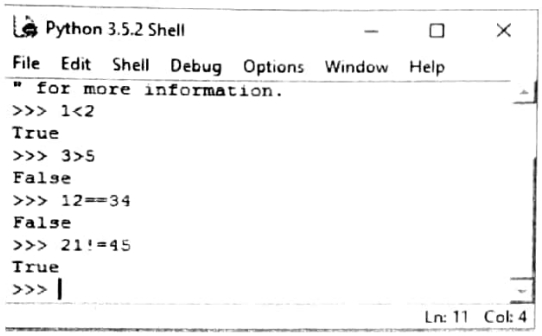


2) **Comparison (Relational) Operators:** These operators compare the values on either sides of them and decide the relation among them. They are also called 'Relational operators'. It either returns 'True' or 'False' according to the condition. The following table illustrates the various types of relational operators are used in python:

Table 6.6: Relational Operators

Operator	Meaning
<	Is less than
<=	Is less than or equal to
>	Is greater than
>=	Is greater than or equal to
= =	Is equal to
!=	Is not equal to

For example, the following code segment illustrates the use of relational operators:



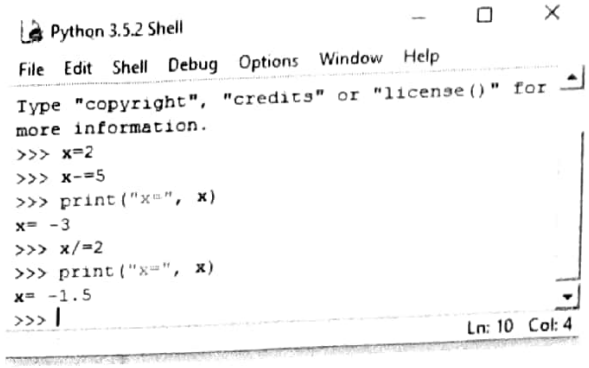
3) **Assignment Operators:** Assignment operators are used in Python to assign values to variables. a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable a on the left. There are various compound operators in Python like a += 5 that adds to the variable and later assigns the same. It is equivalent to a = a + 5.

The following **table 6.7** illustrates the various operators used in python.

Table 6.7: Assignment Operators in Python

Operator	For Example	Equivalent to
=	x = 5	x = 5
+ =	x + = 5	x = x + 5
– =	x – = 5	x = x – 5
* =	x * = 5	x = x * 5
/ =	x / = 5	x = x / 5
% =	x % = 5	x = x % 5
// =	x // = 5	x = x //5
* * =	x ** = 5	x = x ** 5
& =	x & = 5	x = x & 5
| =	x | = 5	x = x | 5
^ =	x ^ = 5	x = x ^ 5
> > =	x >> = 5	x = x >> 5
< < =	x << = 5	x = x << 5

For example, the following code segment illustrates the use of assignment operators:

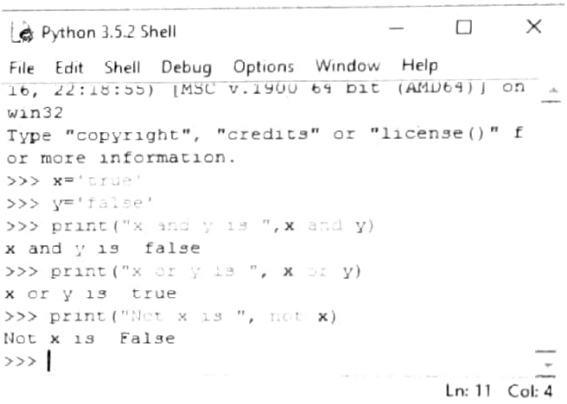


4) **Logical Operators:** There are three logical operators that allow us to build more complex Boolean expressions from simpler Boolean expressions. The following **table 6.8** illustrates three logical operators.

Table 6.8: Logical Operators in Python

Operator	Meaning	For Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

For example, the following code segment illustrates the use of logical operators:



5) **Bitwise Operators:** Bitwise operators act on operands as if they were string of binary digits. It operates bit by bit, hence the name. **For example,** 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

The following **table 6.9** illustrates the various bitwise operations:

Table 6.9: Bitwise Operators in Python

Operator	Meaning	For Example
&	Bitwise AND	x & y = 0(00000000)
|	Bitwise OR	x | y = 14(00001110)
~	Bitwise NOT	~x = –11(11110101)
^	Bitwise XOR	x ^ y = 14(00001110)
>>	Bitwise right shift	x >> 2 = 2(00000010)
<<	Bitwise left shift	x << 2 = 42(00101000)

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 in the following table:

Operator	Meaning	For Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

Note: In a dictionary we can only test for presence of key, not the value.

For example, in the following code segment 'hello' is not present in x (Python is case sensitive). Similarly, 1 is key and 'a' is the value in dictionary y.

```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55
) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more info
rmation.
>>> x='hello world'
>>> y={1:'a', 2:'b'}
>>> print('hello' not in x)
False
>>> print(1 in y)
True
>>>
                                                    Ln: 9  Col: 4
```

7) **Identity Operators:** Identity operators compare the memory locations of two objects. There are two Identity operators 'is' and 'is not'. Two variables that are equal do not imply that they are identical. The following **table 6.10** illustrates the identity operators:

Table 6.10: Identity Operators in Python

Operator	Meaning	For Example
is	True if the operands are identical (refer to the same object)	x is true
is not	True if the operands are not identical (do not refer to the same object)	x is not true

For example, in the following program one can see that x1 and y1 are integers of same values, so they are equal as well as identical. Same is the case with x2 and y2 (strings). But x3 and y3 are list. They are equal but not identical. Since list is mutable (can be changed), interpreter locates them separately in memory although they are equal.

```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 2
2:18:55) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for mo
re information.
>>> x1=2
>>> y1=2
>>> x2='thakur'
>>> y2='thakur'
>>> x3=[1,2]
>>> y3=[1,2]
>>> print(x1 is not y1)
False
>>> print(x2 is y2)
True
>>> print(x3 is y3)
False
>>>
                                                    Ln: 15  Col: 4
```

Ques 24) What do you mean by operator precedence and associativity? Give an example.

Or

Explain the associativity of operators. Also list non-associative operators.

Ans: Operator Precedence

The combination of values, variables, operators and function calls is termed as an expression. Python interpreter can evaluate a valid expression. There can be more than one operator in an expression.

To evaluate these type of expressions there is a rule of precedence in Python. It guides the order in which operation are carried out. **For example,** multiplication has higher precedence than subtraction.

```
>>> 10 – 4 * 2
2
```

But we can change this order using parentheses () as it has higher precedence.

```
>>> (10 – 4) * 2
12
```

The precedence of operators in Python are listed in the following table. It is in descending order, upper group has higher precedence than the lower ones.

Operator Precedence Rule in Python

Operators	Meaning
()	Parentheses
**	Exponent
+x, –x, ~x	Unary plus, Unary minus, Bitwise NOT
*, /, //, %	Multiplication, Division, Floor division Modulus
+, –	Addition, Subtraction
<<, >>	Bitwise shift operators
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
= =, !=, >, >=, <, <=, is, is not, in, not in	Comparisons, Identity, Membership operators
not	Logical NOT
and	Logical AND
or	Logical OR

Associativity

Associativity is the order in which an expression is evaluated that has multiple operator of the same precedence. Almost all the operators have left-to-right associativity. **For example,** multiplication and floor division have the same precedence. Hence, if both of them are present in an expression, left one is evaluates first.

```
>>> 5 * 2 // 3
3
>>> 5 * (2 // 3)
0
```

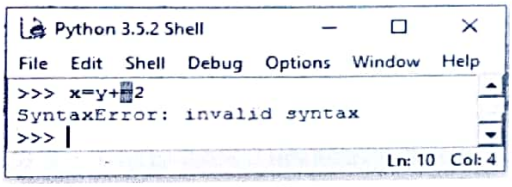
Assignment operator (=) and Exponent operator (**) have right-to-left associativity in Python.

```
>>> 2 ** 3 ** 2
512
>>> (2 ** 3) ** 2
64
```

One can see that 2 ** 3 ** 2 is equivalent to 2 ** (3 ** 2).

Non-Associative Operator

Some operators like assignment operators and comparison operators do not have associativity in Python. There are separate rules for sequences of this kind of operator and cannot be expressed as associativity. **For example,** x < y < z neither means (x < y) < z nor x < (y < z). x < y < z is equivalent to x < y and y < z, and is evaluates from left-to-right. Furthermore, while chaining of assignments like x = y = z is perfectly valid, x = y += z will result into error as shown below.



Ques 25) What is control flow? Give an example.

Or

Explain the control statements/structure used in python.

Or

Define conditional execution and alternative execution.

Ans: Control Flow or Control Statements

It is very important to control the program execution because in real scenarios the situations are full of conditions and if one wants a program to mimic the real world closer then he/she needs to transform those real world situations into his/her program. For this there is the need to control the execution of program statements.

The first word is 'control' that simply means controlling. No programmer wants the default behaviour. Programmer will get the different behaviour by controlling some aspects of the behaviour. Now it comes to 'flow', it is just a way or sequence of program execution.

By default every statement of program is executed one by one in an order they appear in a program code. When we combine the above two words we get 'control flow', that simply means controlling the flow of program execution to get desire behaviour or result. Using control flow one is controlling the statement execution, Now program will no longer be executing in sequence; the execution is controlled by control tools.

For example, in a bank management program one doesn't want to allow the retrieve function to work if the money in an account is zero. In that case he/she needs to skip the retrieve program code and that is control flow.

A script written in python executes each command sequentially from top to bottom. The flow control statements can change this, introducing decision making and repeated execution of code. These statements control the flow of a program's execution based on the results of logical comparisons.

The flow control structures determine whether a block of code has to be executed and how many times. Determining whether a condition is true or false is the basic step to control the flow of the code.

Python provide various tools for flow control. Some of them are if , if .. elif .. else, if..else,while ,for , switch, pass, range, break, else, continue, function etc.

Conditional Execution

In order to write useful programs, we almost always need the ability to check conditions and change the behaviour of the program accordingly. Conditional statements give us this ability. The simplest form is the 'if' statement.

The most common type of flow control statement is the 'if' statement. An 'if' statement's clause (that is, the block following the 'if' statement) will execute if the statement's condition is True. The clause is skipped if the condition is False. In plain English, an 'if' statement could be read as, "If this condition is true, execute the code in the clause." In Python, an 'if' statement is written as:

Syntax:

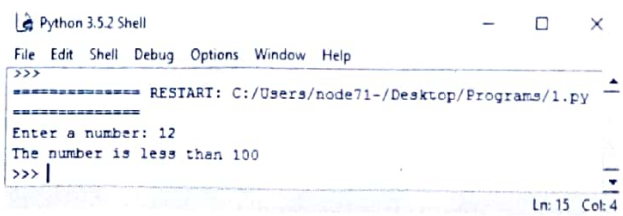
```
if expression:
  do this
```

If the value of expression is true (anything other than zero), do what is written below under the indentation.

For example, consider some number as input and check if the number is less than 100 or not.

```
number = int(input("Enter a number: "))
if number < 100:
  print("The number is less than 100")
```

Output



Alternative Execution/if-else Statement

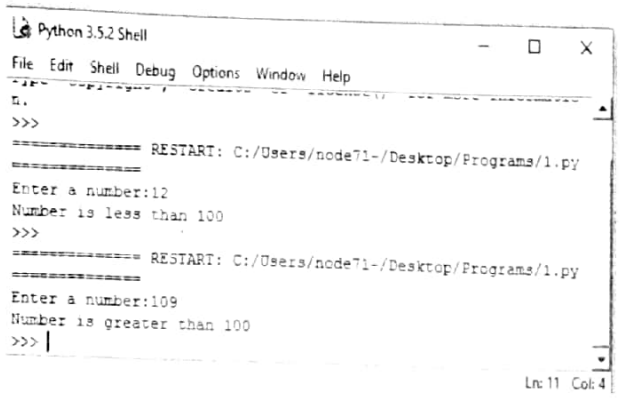
A second form of the 'if' statement is alternative execution, in which there are two possibilities and the condition determines which one gets executed.

When the 'if' statement is not fulfilled, then else statement will be used for alternative execution of the program.

For example, in the above example one wants to print "Greater than" if the number is greater than 100. For this, use the else statement.

```
number=int(input("Enter a number:"))
if number<100:
    print("Number is less than 100")
else:
    print("Number is greater than 100")
```

Output



Ques 26) What do you mean by iteration?

Or

What is looping in python? How it can be achieved? Give an example.

Or

How while statement is differ from if statement?

Ans: Iteration/ Looping

When a process or sequence in a computer program is repeated, this is 'iteration'. In a computer program, a common form of iterations is a **loop**, which repeats code to determine values for multiple variables or sometimes just a single variable (adding up multiple values together).

Because iteration is so common, Python provides several language features to make it easier. Two form of iteration provides by python are 'for' statement and 'while' statement.

for Loop Statement

The 'for' statement in Python has the ability to iterate over the items of any sequence, such as a list or a string.

The while loop keeps looping while its condition is True (which is the reason for its name), but what if one wants to execute a block of code only a certain number of times, he/she can do this with a for loop statement and the range() function.

Syntax:

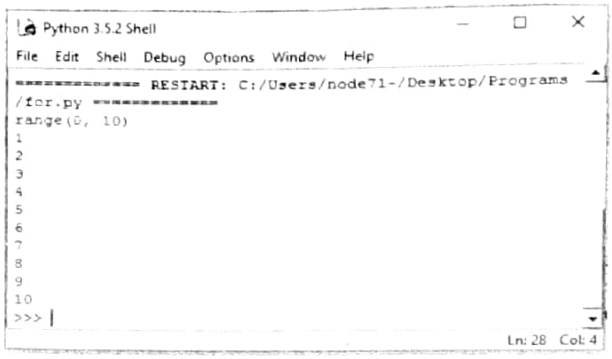
```
for iterating_var in sequence:
    statements(s)
```

If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable iterating_var. Next, the statements block is executed. Each item in the list is assigned to iterating_var, and the statement(s) block is executed until the entire sequence is exhausted.

For example, consider the following code segment:

```
counter = range(10)
print(counter)

for i in counter:
    print(i + 1)
```

Output



In the above program an array that contains the numbers from 0 up to 9 is created by counter = range(10). That array is used in the 'for' statement which executes the code 10 times, each time assigning to the variable i one entry from the array, then adds 1 and prints out the result.

range() Function

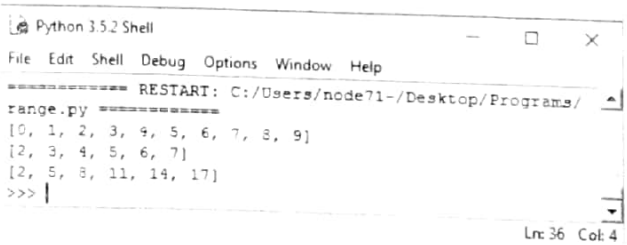
One can generate a sequence of numbers using range() function, **for example,** range(10) will generate numbers from 0 to 9 (10 numbers). One can also define the start, stop and step size as **range(start, stop, step size)**. Step size defaults to 1 if not provided.

This function does not store all the values in memory, it would be inefficient. So, it remembers the start, stop, step size and generates the next number on the go. To force this function to output all the items, one can use the function **list()**.

The following segment of code will illustrate this concept:

```
range(10)
range(0, 10)
print(list(range(10)))
print(list(range(2,8)))
print(list(range(2,20,3)))
```

Output



range() function can be used in for loops to iterate through a sequence of numbers as shown in the for loop example.

while Loop Statement

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

Syntax:

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

Here, statement(s) may be a single statement or a block of statements with uniform indent. 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.

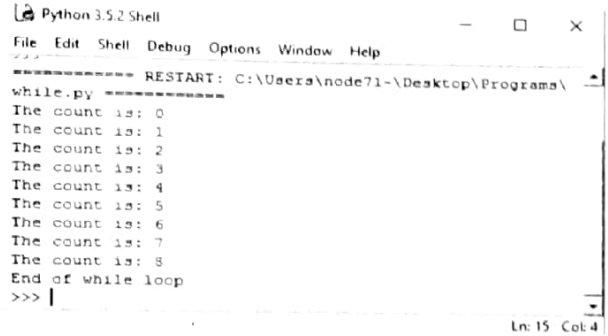
In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

The key point of the while loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

For example, consider the following code segment:

```
count = 0
while (count < 9):
  print ('The count is:', count)
  count = count + 1

print ("End of while loop")
```

Output



In the above while loop **program**, the condition of 'count < 9' is always checked at the start of each iteration (that is, each time the loop is executed). If the condition is True, then the clause is executed and print the message, and afterward, the condition is checked again. The while loop will be skip, if the condition is found to be False.

Difference between while and if Statement

One can see that a 'while' statement looks similar to an 'if' statement. The difference is in how they behave. At the end of an 'if' statement, the program execution continues after the 'if' statement. But at the end of a 'while' loop, the program execution jumps back to the start of the while statement.

Ques 27) Write the program to find the largest number among the three input numbers.

Ans: Program to Find the Largest Number among the Three Input Numbers

```
# Store three numbers from user
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
num3 = float(input("Enter third number: "))

# Find largest number
if (num1 > num2) and (num1 > num3):
  largest = num1
elif (num2 > num1) and (num2 > num3):
  largest = num2
else:
  largest = num3

# Display the sum
print("The largest number is", largest)
```

Output

FUNCTIONS IN PYTHON

Ques 28) Explain the function. Write the syntax of writing the function in python.

Why do we need functions? What are the advantages of using a function? (Jan 2016 [3])

Or

How function can be defined in python? Also discuss their types.

Ans: Function

In Python, function is a group of related statements that perform a specific task. Functions help break program into smaller and modular chucks. As program grows larger and larger, functions make it more organised and manageable. Furthermore, it avoids repetition and makes code reusable.

A function is a block of organised, reusable code that is used to perform a single, related action. Functions provide better modularity for application and a high degree of code reusing.

Defining a Function

One can define functions to provide the required functionality. Some of the rules to define a function in Python are as follows:

1) One needs to write the declaration part of the function. In Python, it starts with a keyword 'def' and then the function name.
2) Function in Python can take argument(s) or it can be without any argument. Arguments are separated by commas within the opening and closing parentheses of the function. Function name ends with a colon (:).
3) After the declaration part, program statements are written and they should be indented.
4) Functions can or cannot have a return statement.

Syntax:

```
def function_name( parameter_list ):
    Statements, i.e. the function body
return [expression]
```

For example, consider the following code segment of function that prints "Thakur Publishers".

```
def Func():
  print("Thakur Publishers ")
```

The function is called **Func** and the first two lines only define it, python will do nothing unless the function is used inside the program.

Let us consider another **example** in which function takes a string as input parameter and prints it on standard screen.

```
def printme(str):
  "This prints a passed string into this function"
  print str
 return
```

Types of Functions

There are two basic types of functions:

1) **Built-in Functions:** The built-in functions are part of the Python language. **For example,** dir(), len() and abs() are the built-in functions of python.
2) **User-Defined Functions:** The user defined functions are functions created by the users with the def keyword.

Need of Functions

1) A common usage of functions in computer languages is to implement mathematical functions. Such a function is computing one or more results, which are entirely determined by the parameters passed to it.
2) In the most general sense, a function is a structuring element in programming languages to group a set of statements so they can be utilised more than once in a program. The only way to accomplish this without functions would be to reuse code by copying it and adapt it to its different context.
3) Using functions usually enhances the comprehensibility and quality of the program.
4) It also lowers the cost for development and maintenance of the software.

Advantages of Functions

1) **Decomposing Complex Problems into Simpler Pieces:** Users can split large programming code into small logical chunks and put it into blocks, which are nothing but user-defined functions. It helps to read, maintain and debug application code in a better way.
2) **Reducing Duplication of Code:** User-defined functions are very useful for repetitive code usage. It reduces development time and effort.
3) **Reuse of Code:** User-defined functions are re-usable components. So it can be used across application code.
4) **Improving Clarity and Modification of the Code:** These functions can be modified or changed as per requirement, without changing the application code.

Ques 29) How functions can be called in the program? Write the syntax for function call.

Or

What do you mean by function calling? Explain with suitable example.

Or

Explain how dot notation is used to calling a function.

Ans: Function Calling

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.

Once the basic structure of a function is finalised, one can executes it by calling it from another function or directly from the Python prompt. If anyone calls a function, the statements inside the function body are executed. They are not executed until the function is called. To call a function one simply types the function name with appropriate parameters.

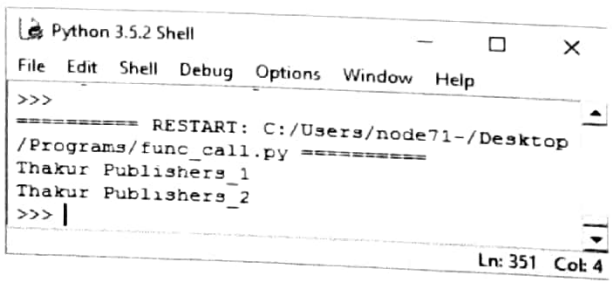
Syntax:

```
func_name()
```

For example, the following code segment illustrates the function call:

```
# Function definition is here
def func( str ):
  "This prints a passed string into this function"
  print (str)
  return;

# Now one can call func function
func("Thakur Publishers_1")
func("Thakur Publishers_2")
```

Output



Using Dot Notation to Calling a Function

The dot notation is used for calling a function in another module by specifying the module name followed by a dot (period) and the function name.

Syntax:

Module_name.Function_name

For example, there are some functions that reside in modules that have to be loaded before one can use them like string, math, and random. User loads these modules by using the **import** directive.

```
import string
import math, random
```

After loading the module one can call on the functions using the **dot** operator as follows:

```
#!/usr/bin/python
import math, random
x = 7
y = math.sqrt(x)
z = random.random()
print(y)
print(z)
```

Output

```
2.6457513110645907
0.6381911268272096
```

Ques 30) Write a function that interchanges the value of two integers a and b without using any extra variable.

Ans: Function to Swap A and B Without Using Third Variable

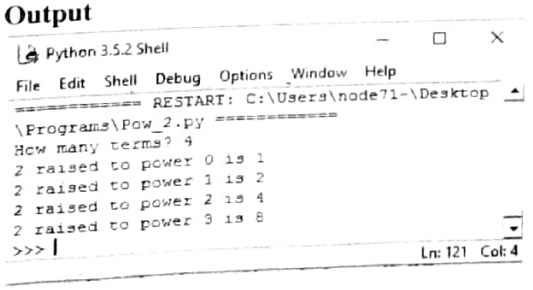
```
def swap(a, b):
    a = a + b
    b = a – b
    a = a – b
    print "After swapping the first value is –>", a
    print "After swapping the second value is –>", b
```

Ques 31) Write a Python program to display the powers of 2 using anonymous function.

Ans: Program to Display the Powers of 2 using Anonymous Function

```
# Take number of terms from user
terms = int(input("How many terms? "))

# use anonymous function
result = list(map(lambda x: 2 ** x, range(terms)))

# display the result
for i in range(terms):
  print("2 raised to power",i,"is",result[i])
```

Output

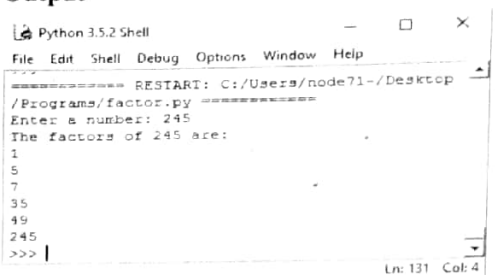


Ques 32) Write a Python program to find the factors of a number.

Ans: Program to Find the Factors of a Number

```
# define a function
def print_factors(x):
  """This function takes a
  number and prints the factors"""

  print("The factors of",x,"are:")
  for i in range(1, x + 1):
    if x % i == 0:
      print(i)

# take input from the user
num = int(input("Enter a number: "))

print_factors(num)
```

Output

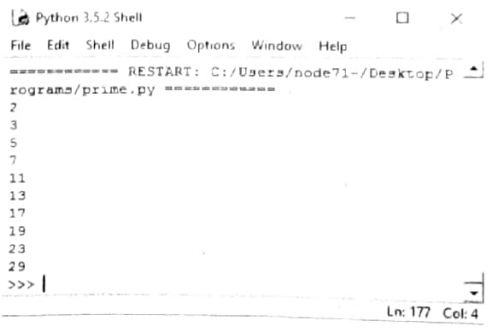


Ques 33) Write a Python program to make a prime number list from (1, 30).

Ans: Program to Make a Prime Number List from (1, 30)

```
import math
def isPrime(n):
  if n <= 1:
    return False
  if n == 2:
    return True
  if n % 2 == 0:
    return False
  for t in range(3, int(math.sqrt(n)+1),2):
    if n % t == 0:
      return False
  return True
for n in range(30):
  if isPrime(n):
    print (n )
```

Output

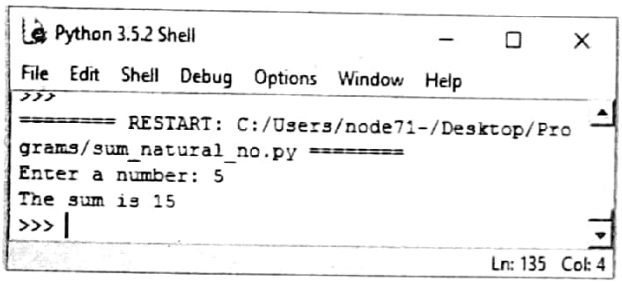


Ques 34) Write a Python program to find the sum of natural numbers up to n using recursive function.

Ans: Program to Find the Sum of Natural Numbers up to n using Recursive Function

```
def recur_sum(n):
  """Function to return the sum
  of natural numbers using recursion"""
  if n <= 1:
    return n
  else:
    return n + recur_sum(n-1)

# take input from the user
num = int(input("Enter a number: "))

if num < 0:
  print("Enter a positive number")
else:
  print("The sum is",recur_sum(num))
```

Output

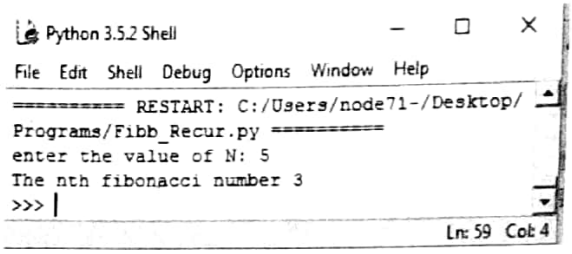


Ques 35) Write a Python program to compute the nth Fibonacci number. Use a recursive function for the implementation.

Ans: Program to Compute the nth Fibonacci Number using a Recursive Function

```
def recur_fibo(n):
#Recursive function to print Fibonacci number
  if n <= 1:
    return n
  else:
    return(recur_fibo(n-1) + recur_fibo(n-2))

# take input from the user
nterms = int(input("enter the value of N: "))
# check if the number of terms is valid
if nterms <= 0:
  print("Plese enter a positive integer")
else:
  for i in range(nterms):
    f= recur_fibo(i)
  print("The nth fibonacci number",f)
```

Output



Ques 36) Write a python program to find sum of digits of a number (preferably a recursion function).

Ans: Program to Find Sum of Digits of a Number using Recursion

```
Sum = 0
def Sum_Of_Digits(Number):
  global Sum
  if(Number > 0):
    Reminder = Number % 10
    Sum = Sum + Reminder
    Sum_Of_Digits(Number //10)
  return Sum

Number = int(input("Please Enter any Number: "))
Sum = Sum_Of_Digits(Number)
print("\nSum of the digits of Given Number = %d" %Sum)
```

Output