Answer Key Question paper format - B.Tech I Year (IE 2)

Programming For Problem Solving (SRM Institute of Science and Technology)

Registration Number

SRM Institute of Science and Technology
Department of Computer Science & Engineering
Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY (Deemed to be University u/s 3 of UGC Act, 1956) DELHI-NCR CAMPUS, GHAZIABAD (U.P)

Academic Year: 2023-24 (ODD)

Test : Internal Examination II — **Date & Session : 28/11/2023 & FN**
Course Code & Title : 21CSS101J & Programming for Problem Solving — **Duration: 2.5 Hours**
Year & Sem : I Year & I Semester — **Max. Marks: 75**

Part - A

Answer all questions — **(10 x 1 = 10 Marks)**

Q. No	Question	Marks	BL	CO	PO
1	What will be the outcome of this python expression 4+3%5? **(a) 7** (b) 2 (c) 4 (d) 1	1	L3	4	3
2	Which of the following is used to give single line comment in python: (a) // **(b) #** (c) ! (d) /*	1	L3	4	2
3	How do we define a block of code in python? (a) Curly braces **(b) Indentation** (c) # (d) Key	1	L2	4	3
4	Which of the following functions is a built-in function in python? (a) factorial() **(b) print()** (c) seed() (d) sqrt()	1	L2	4	2
5	Which of the following is not a core data type in python? (a) Tuple **(b) Void** (c) List (d) Dictionary	1	L2	4	3
6	What of these is not a correct parameter passing method in c? **(a)Call by record** (b) Call by value (c)Call by reference (d) Call by passing address	1	L3	3	3
7	The scope of function is limited to? (a) Only function (b) Current block **(c) Whole file** (d) Directory	1	L3	3	3
8	_____ is a pointer that occurs at the time when the object is deallocated from memory without modifying the value of the pointer. **(a) Dangling pointer** (b) Wild Pointer (c) void pointer (d) NULL pointer	1	L2	3	3
9	What is the output of this statement "c=printf("HELLO")"? (a) Error (b) HELLO **(c) HELLO 5** (d) 5	1	L3	3	2
10	Which one of the following marks the end of a string in C? **(a) '\x0'** (b) EOF (c) EOS (d) null	1	L2	3	3

Part B

Answer any seven questions — **5M x 7Q = 35 Marks**

11	Differentiate between call by value and call by reference with examples. Answer:	5	L3	3	4

Call by value in C

In call by value method, the value of the actual parameters is copied into the formal parameters. In other words, we can say that the value of the variable is used in the function call in the call by value method.

In call by value method, we can not modify the value of the actual parameter by the formal parameter.

In call by value, different memory is allocated for actual and formal parameters since the value of the actual parameter is copied into the formal parameter.

The actual parameter is the argument which is used in the function call whereas formal parameter is the argument which is used in the function definition.

Let's try to understand the concept of call by value in c language by the example given below:

```
#include<stdio.h>
void change(int num) {
   printf("Before adding value inside function num=%d \n",num);
   num=num+100;
   printf("After adding value inside function num=%d \n", num);
}
int main() {
   int x=100;
   printf("Before function call x=%d \n", x);
   change(x);//passing value in function
   printf("After function call x=%d \n", x);
return 0;
}
```

Output

```
Before function call x=100
Before adding value inside function num=100
After adding value inside function num=200
After function call x=100
```

Call by Value Example: Swapping the values of the two variables

```
#include <stdio.h>
void swap(int , int); //prototype of the function
int main()
{
   int a = 10;
   int b = 20;
   printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing
the value of a and b in main
   swap(a,b);
   printf("After swapping values in main a = %d, b = %d\n",a,b); // The value of
actual parameters do not change by changing the formal parameters in call by
value, a = 10, b = 20
}
void swap (int a, int b)
{
   int temp;
   temp = a;
   a=b;
   b=temp;
   printf("After swapping values in function a = %d, b = %d\n",a,b); // Formal
parameters, a = 20, b = 10
}
```

Output

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
```

After swapping values in main a = 10, b = 20
Call by reference in C
In call by reference, the address of the variable is passed into the function call as the actual parameter.
The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed.
In call by reference, the memory allocation is similar for both formal parameters and actual parameters. All the operations in the function are performed on the value stored at the address of the actual parameters, and the modified value gets stored at the same address.
Consider the following example for the call by reference.

```
#include<stdio.h>
void change(int *num) {
   printf("Before adding value inside function num=%d \n",*num);
   (*num) += 100;
   printf("After adding value inside function num=%d \n", *num);
}
int main() {
   int x=100;
   printf("Before function call x=%d \n", x);
   change(&x);//passing reference in function
   printf("After function call x=%d \n", x);
return 0;
}
```

Output

```
Before function call x=100
Before adding value inside function num=100
After adding value inside function num=200
After function call x=200
```

Call by reference Example: Swapping the values of the two variables

```
#include <stdio.h>
void swap(int *, int *); //prototype of the function
int main()
{
   int a = 10;
   int b = 20;
   printf("Before swapping the values in main a = %d, b = %d\n",a,b);
 // printing the value of a and b in main
   swap(&a,&b);
   printf("After swapping values in main a = %d, b = %d\n",a,b);
// The values of actual parameters do change in call by reference, a = 10, b = 20
}
void swap (int *a, int *b)
{
   int temp;
   temp = *a;
   *a=*b;
   *b=temp;
   printf("After swapping values in function a = %d, b = %d\n",*a,*b);
// Formal parameters, a = 20, b = 10
}
```

Output

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
```

After swapping values in main a = 20, b = 10

12 Write the syntax and an example of sprintf, sscanf, strrev, and strcpy. 5 L3 3 4

Syntax:
int sprintf(char *str, const char *string,...);

```
// Example program to demonstrate sprintf()
#include <stdio.h>
int main()
{
   char buffer[50];
   int a = 10, b = 20, c;
   c = a + b;
   sprintf(buffer, "Sum of %d and %d is %d", a, b, c);

   // The string "sum of 10 and 20 is 30" is stored
   // into buffer instead of printing on stdout
   printf("%s", buffer);

   return 0;
}
```

Output
Sum of 10 and 20 is 30

The C library function int sscanf(const char *str, const char *format, ...) reads formatted input from a string.

Declaration
Following is the declaration for sscanf() function.

int sscanf(const char *str, const char *format, ...)

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main () {
  int day, year;
  char weekday[20], month[20], dtm[100];

  strcpy( dtm, "Saturday March 25 1989" );
  sscanf( dtm, "%s %s %d  %d", weekday, month, &day, &year );

  printf("%s %d, %d = %s\n", month, day, year, weekday );

  return(0);
}
```

Output:
March 25, 1989 = Saturday

The strrev() function is a built-in function in C and is defined in string.h header file. The strrev() function is used to reverse the given string.

Syntax:

```
char *strrev(char *str);
```

Parameter:

str: The given string which is needed to be reversed.

```
// C program to demonstrate
// example of strrev() function

#include <stdio.h>
#include <string.h>

int main()
{
        char str[50] = "geeksforgeeks";

        printf("The given string is =%s\n", str);

        printf("After reversing string is =%s", strrev(str));

        return 0;
}
```

Output:

```
The given string is = geeksforgeeks
After reversing string is = skeegrofskeeg
```

The function prototype of strcpy() is:

```
char* strcpy(char* destination, const char* source);
```

The strcpy() function copies the string pointed by source (including the null character) to the destination.

The strcpy() function also returns the copied string.
The strcpy() function is defined in the string.h header file.

Example: strcpy()

```
#include <stdio.h>
#include <string.h>

int main() {
  char str1[20] = "C programming";
  char str2[20];

  // copying str1 to str2
  strcpy(str2, str1);

  puts(str2); // C programming

  return 0;
}
```

Output

C programming

13 Write a code in Python to check if the year is a leap year or not. 5 L3 4 5

```
# Python Program to Check Leap Year

year=int(input("Enter year to be checked:"))
if(year%4==0 and year%100!=0 or year%400==0):
   print("The year is a leap year!")
else:
   print("The year isn't a leap year!")
```

14 Write a code in C to check if the string entered is a Palindrome using functions. 5 L3 3 4

```
int checkPalindrome(int number)
{
 // declare variables
 int temp, remainder, rev=0;

 // copy of original number
 temp = number;

 // loop to repeat
 while( number!=0 )
 {
   // find last digit
   remainder = number % 10;

   // calculate reverse
   rev = rev*10 + remainder;

   // remove last digit
   number /= 10;
 }

 /* if reverse is equal to the
  * original number then it is a
  * palindrome number else it is not.
  */
 if ( rev == temp ) return 0;
 else return 1;
}
```

15 How do we create series objects and data frame objects in pandas? Give examples. 5 L3 5 6

Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called index. Labels need not be unique but must be a hashable type. The object supports both integer and label-based indexing and provides a host of methods for performing operations involving the index.

To create Series with any of the methods make sure to import pandas library.

Creating an empty Series: Series() function of Pandas is used to create a series. A

basic series, which can be created is an Empty Series.

```
# import pandas as pd
import pandas as pd

# Creating empty series
ser = pd.Series()

print(ser)
```

Output :

```
Series([], dtype: float64)
```

By default, the data type of Series is float.

Creating a series from array: In order to create a series from NumPy array, we have to import numpy module and have to use array() function.

```
# import pandas as pd
import pandas as pd

# import numpy as np
import numpy as np

# simple array
data = np.array(['g', 'e', 'e', 'k', 's'])

ser = pd.Series(data)
print(ser)
```

By default, the index of the series starts from 0 till the length of series -1.

Creating a series from array with an index: In order to create a series by explicitly proving index instead of the default, we have to provide a list of elements to the index parameter with the same number of elements as it is an array.

```
# import pandas as pd
import pandas as pd

# import numpy as np
import numpy as np

# simple array
data = np.array(['g', 'e', 'e', 'k', 's'])

# providing an index
ser = pd.Series(data, index=[10, 11, 12, 13, 14])
print(ser)
```

Creating a series from Lists: In order to create a series from list, we have to first create a list after that we can create a series from list.

```
import pandas as pd

# a simple list
list = ['g', 'e', 'e', 'k', 's']
```

```
# create series form a list
ser = pd.Series(list)
print(ser)
```

Creating a series from Dictionary: In order to create a series from the dictionary, we have to first create a dictionary after that we can make a series using dictionary. Dictionary keys are used to construct indexes of Series.

```
import pandas as pd

# a simple dictionary
dict = {'Geeks': 10,
                    'for': 20,
                    'geeks': 30}

# create series from dictionary
ser = pd.Series(dict)

print(ser)
```

Creating a series from Scalar value: In order to create a series from scalar value, an index must be provided. The scalar value will be repeated to match the length of the index.

```
import pandas as pd

import numpy as np

# giving a scalar value with index
ser = pd.Series(10, index=[0, 1, 2, 3, 4, 5])

print(ser)
```

Creating a Series using range function:

```
# code
import pandas as pd
ser=pd.Series(range(10))
print(ser)
```

Creating a Series using for loop and list comprehension:

```
import pandas as pd
ser=pd.Series(range(1,20,3), index=[x for x in 'abcdefg'])
print(ser)
```

Series is a type of list in Pandas that can take integer values, string values, double values, and more. But in Pandas Series we return an object in the form of a list, having an index starting from 0 to n, Where n is the length of values in the series. Later in this article, we will discuss Dataframes in pandas, but we first need to understand the main difference between Series and Dataframe.

Series can only contain a single list with an index, whereas Dataframe can be made of more than one series or we can say that a Dataframe is a collection of series that can be used to analyze the data.

Creating Pandas DataFrames from Series

```
# importing pandas library
```

```
import pandas as pd

# Creating a list
author = ['Jitender', 'Purnima',
                'Arpit', 'Jyoti']
# Creating a Series by passing list
# variable to Series() function
auth_series = pd.Series(author)
# Printing Series
print(auth_series)
```

Let’s check the type of Series:

```
print(type(auth_series))
```

Create DataFrame From Multiple Series

We have created two lists ‘author’ and article’ which have been passed to pd.Series() functions to create two Series. After creating the Series, we created a dictionary and passed Series objects as values of the dictionary, and the keys of the dictionary will be served as Columns of the Dataframe.

```
# Importing Pandas library
import pandas as pd

# Creating two lists
author = ['Jitender', 'Purnima',
                'Arpit', 'Jyoti']
article = [210, 211, 114, 178]

# Creating two Series by passing lists
auth_series = pd.Series(author)
article_series = pd.Series(article)

# Creating a dictionary by passing Series objects as values
frame = {'Author': auth_series,
                'Article': article_series}

# Creating DataFrame by passing Dictionary
result = pd.DataFrame(frame)

# Printing elements of Dataframe
print(result)
```

Add a Column in Pandas Dataframe

We have added one more series externally named as the age of the authors, then directly added this series in the Pandas Dataframe.

```
# Importing pandas library
import pandas as pd
# Creating Series
auth_series = pd.Series(['Jitender',
                'Purnima', 'Arpit', 'Jyoti'])
article_series = pd.Series([210, 211, 114, 178])
# Creating Dictionary
frame = {'Author': auth_series,
                'Article': article_series}
# Creating Dataframe
result = pd.DataFrame(frame)
```

```
# Creating another list
age = [21, 21, 24, 23]
# Creating new column in the dataframe by
# providing s Series created using list
result['Age'] = pd.Series(age)
# Printing dataframe
print(result)
```

Missing value in Pandas Dataframe
Remember one thing if any value is missing then by default it will be converted into NaN value, i.e, null by default.

```
# Importing pandas library
import pandas as pd
# Creating Series
auth_series = pd.Series(['Jitender',
                         'Purnima', 'Arpit', 'Jyoti'])
article_series = pd.Series([210, 211, 114, 178])
# Creating Dictionary
frame = {'Author': auth_series,
         'Article': article_series}
# Creating Dataframe
result = pd.DataFrame(frame)
# Creating another list
age = [21, 21, 24]
# Creating new column in the dataframe by
# providing s Series created using list
result['Age'] = pd.Series(age)
# Printing dataframe
print(result)
```

Creating a Dataframe using a dictionary of Series
Here, we have passed a dictionary that has been created using a series as values then passed this dictionary to create a Dataframe. We can see while creating a Dataframe using Python Dictionary, the keys of the dictionary will become Columns and values will become Rows.

```
# Importing pandas library
import pandas as pd
# Creating dictionary of Series
dict1 = {'Auth_Name': pd.Series(['Jitender',
                                 'Purnima', 'Arpit', 'Jyoti']),
         'Author_Book_No':\
                                 pd.Series([210, 211, 114, 178]),
         'Age': pd.Series([21, 21, 24, 23])}

# Creating Dataframe
df = pd.DataFrame(dict1)
# Printing dataframe
print(df)
```

Explicit Indexing in Pandas Dataframe
Here we can see after providing an index to the dataframe explicitly, it has filled all data with NaN values since we have created this dataframe using Series and Series has its own default indices(0,1,2) which is why when indices of both dataframe and Series do not match, we got all NaN values.

```
# Importing pandas library
```

```
import pandas as pd
# Creating dictionary of Series
dict1 = {'Auth_Name': pd.Series(['Jitender',
                                 'Purnima', 'Arpit', 'Jyoti']),
         'Author_Book_No': pd.Series([210, 211, 114, 178]),
         'Age': pd.Series([21, 21, 24, 23])}

# Creating Dataframe
df = pd.DataFrame(dict1, index=['SNo1', 'SNo2', 'SNo3', 'SNo4'])
# Printing dataframe
print(df)
```

Here, we can rectify this problem by providing the same index values to every Series element.

```
# This code is provided by Sheetal Verma
# Importing pandas library
import pandas as pd
# Creating dictionary of Series
dict1 = {'Auth_Name': pd.Series(['Jitender',
                                 'Purnima', 'Arpit', 'Jyoti'],
                                index=['SNo1', 'SNo2', 'SNo3', 'SNo4']),
                    'Author_Book_No': pd.Series([210, 211, 114, 178],
                                                index=['SNo1', 'SNo2', 'SNo3', 'SNo4']),
                    'Age': pd.Series([21, 21, 24, 23],
                                     index=['SNo1', 'SNo2', 'SNo3', 'SNo4'])}

# Creating Dataframe
df = pd.DataFrame(dict1, index=['SNo1', 'SNo2', 'SNo3', 'SNo4'])
# Printing dataframe
print(df)
```

16 Write a short note on Numpy array attributes with suitable examples. 5 L3 5 6

NumPy Array Attributes
In NumPy, attributes are properties of NumPy arrays that provide information about the array's shape, size, data type, dimension, and so on.

For example, to get the dimension of an array, we can use the ndim attribute.

There are numerous attributes available in NumPy, which we'll learn below.

Common NumPy Attributes
Here are some of the commonly used NumPy attributes:

Attributes	Description
ndim	returns number of dimension of the array
size	returns number of elements in the array
dtype	returns data type of elements in the array
shape	returns the size of the array in each dimension.
itemsize	returns the size (in bytes) of each elements in the array
data	returns the buffer containing actual elements of the array in memory

To access the Numpy attributes, we use the . notation. For example,

array1.ndim
This returns the number of dimensions in array1.

Numpy Array ndim Attribute
The ndim attribute returns the number of dimensions in the numpy array. For example,

```
import numpy as np

# create a 2-D array
array1 = np.array([[2, 4, 6],
                   [1, 3, 5]])

# check the dimension of array1
print(array1.ndim)

# Output: 2
```

Run Code
In this example, array1.ndim returns the number of dimensions present in array1. As array1 is a 2D array, we got 2 as an output.

NumPy Array size Attribute
The size attribute returns the total number of elements in the given array.

Let's see an example.

```
import numpy as np

array1 = np.array([[1, 2, 3],
                   [6, 7, 8]])

# return total number of elements in array1
print(array1.size)

# Output: 6
```

Run Code
In this example, array1.size returns the total number of elements in the array1 array, regardless of the number of dimensions.

Since these are a total of 6 elements in array1, the size attribute returns 6.

NumPy Array shape Attribute
In NumPy, the shape attribute returns a tuple of integers that gives the size of the array in each dimension. For example,

```
import numpy as np

array1 = np.array([[1, 2, 3],
                   [6, 7, 8]])

# return a tuple that gives size of array in each dimension
print(array1.shape)

# Output: (2,3)
```

Run Code
Here, array1 is a 2-D array that has 2 rows and 3 columns. So array1.shape returns the tuple (2,3) as an output.

NumPy Array dtype Attribute
We can use the dtype attribute to check the datatype of a NumPy array. For

example,

```
import numpy as np

# create an array of integers
array1 = np.array([6, 7, 8])

# check the data type of array1
print(array1.dtype)

# Output: int64
```

Run Code

In the above example, the dtype attribute returns the data type of array1.

Since array1 is an array of integers, the data type of array1 is inferred as int64 by default.

Note: To learn more about the dtype attribute to check the datatype of an array, visit NumPy Data Types.

NumPy Array itemsize Attribute

In NumPy, the itemsize attribute determines size (in bytes) of each element in the array. For example,

```
import numpy as np

# create a default 1-D array of integers
array1 = np.array([6, 7, 8, 10, 13])

# create a 1-D array of 32-bit integers
array2 = np.array([6, 7, 8, 10, 13], dtype=np.int32)

# use of itemsize to determine size of each array element of array1 and array2
print(array1.itemsize)  # prints 8
print(array2.itemsize)  # prints 4
```

Run Code

Output

```
8
4
```

Here,

array1 is an array containing 64-bit integers by default, which uses 8 bytes of memory per element. So, itemsize returns 8 as the size of each element.
array2 is an array of 32-bit integers, so each element in this array uses only 4 bytes of memory. So, itemsize returns 4 as the size of each element.

NumPy Array data Attribute

In NumPy, we can get a buffer containing actual elements of the array in memory using the data attribute.

In simpler terms, the data attribute is like a pointer to the memory location where the array's data is stored in the computer's memory.

Let's see an example.

```
import numpy as np
```

```
array1 = np.array([6, 7, 8])
array2 = np.array([[1, 2, 3],
                   [6, 7, 8]])

# print memory address of array1's and array2's data
print("\nData of array1 is: ",array1.data)
print("Data of array2 is: ",array2.data)
```
Run Code
Output

```
Data of array1 is: <memory at 0x7f746fea4a00>
Data of array2 is:  <memory at 0x7f746ff6a5a0>
```
Here, the data attribute returns the memory addresses of the data for array1 and array2 respectively.

17 Discuss the datatypes used in Python with examples. 5 L3 4 5

Built-in Data Types
In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview
None Type:	NoneType

Getting the Data Type
You can get the data type of any object by using the type() function:

ExampleGet your own Python Server
Print the data type of the variable x:

```
x = 5
print(type(x))
```
Setting the Data Type
In Python, the data type is set when you assign a value to a variable:

Example	Data Type
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name" : "John", "age" : 36}	dict
x = {"apple", "banana", "cherry"}	set
x = frozenset({"apple", "banana", "cherry"})	frozenset
x = True	bool
x = b"Hello"	bytes
x = bytearray(5)	bytearray

	x = memoryview(bytes(5)) memoryview x = None NoneType				
18	Discuss with examples few methods in a list. Python has a set of built-in methods that you can use on lists/arrays. Method Description append() Adds an element at the end of the list clear() Removes all the elements from the list copy() Returns a copy of the list count() Returns the number of elements with the specified value extend() Add the elements of a list (or any iterable), to the end of the current list index() Returns the index of the first element with the specified value insert() Adds an element at the specified position pop() Removes the element at the specified position remove() Removes the first item with the specified value reverse() Reverses the order of the list sort() Sorts the list	5	L3	4	5

19	Write a function in C to print the largest integer in an array of 10 numbers
	#include<stdio.h>
	void main(void)
	{
	int a[10]={4,2,5,23,65,26,6,34,21,1};
	int largest(int x[]);
	printf("Largest of 10 numbers is %d\n",largest(a));
	}
	int largest(int x[])
	{
	int l=x[0];
	for(int i=1;i<10;i++)
	{
	if(l<x[i])
	l=x[i];
	}
	return l;
	}
		5	L3	3	4

Part C

Answer all questions **10M x 3Q = 30 Marks**

20	(A) How is exception handling done in Python? Give examples. The try block lets you test a block of code for errors. The except block lets you handle the error. The else block lets you execute code when there is no error. The finally block lets you execute code, regardless of the result of the try- and except blocks. Exception Handling When an error occurs, or exception as we call it, Python will normally stop and generate an error message. These exceptions can be handled using the try statement: The try block will generate an exception, because x is not defined:	10	L3	4	5

```
try:
  print(x)
except:
  print("An exception occurred")
```

The finally block, if specified, will be executed regardless if the try block raises an error or not.

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

This can be useful to close objects and clean up resources:
Raise an exception
As a Python developer you can choose to throw an exception if a condition occurs.

To throw (or raise) an exception, use the raise keyword.

```
x = -1
if x < 0:
  raise Exception("Sorry, no numbers below zero")
```

(OR)

(B) Write a code in C to check if two strings are an anagram or not. Also count the frequency of each alphabet in the string. 10 L3 3 5

```
#include<stdio.h>

int main() {
   char str[1000];
   int i;

   printf("Enter a string of lowercase English letters : ");
   fgets(str, sizeof(str), stdin);

   // Step 1 : Creating a freq[] array.
   int freq[26] = {0};

   // Step 2: Traversing the string.
   for (i = 0; str[i] != '\0'; i++) {
      // Increment character count at the ith position by 1.
      freq[str[i]-'a'] += 1;
   }

   // Step 3 : Traversing the freq[] array to print character.
   printf("The frequency of characters is -\n");
   for (i = 0; i < 26; i++) {

      if (freq[i] != 0) {

         // Here, we can obtain characters by adding the ASCII value of 'a',
ASCII(b) = ASCII(a) + 1.
```

```
            char char_ = 'a' + i;
            printf("\t%c = %d\n", char_, freq[i]);
        }
    }

    return 0;
}
```

21 (A) Differentiate between numpy and pandas. 10 L3 5 6

Criteria	Pandas	NumPy
Fundamental Data Object	Series and DataFrames	N-dimensional array or ndarray
Memory Consumption	More	Less
Performance on smaller datasets	Slower	Faster
Performance on larger datasets	Faster	Slower
Data Object Type	Heterogeneous	Homogeneous
Access Methods	Index positions and index labels	Index positions
Indexing	Slower	Faster

(OR)

(B) What are the different ways of looping in Python? Give examples. 10 L3 4 6

Python programming language provides the following types of loops to handle looping requirements. Python provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition-checking time.

Types of Python Loops

While Loop in Python
For Loop in Python

While Loop in Python
In Python, a while loop is used to execute a block of statements

repeatedly until a given condition is satisfied. And when the condition becomes false, the line immediately after the loop in the program is executed.

While Loop Syntax:

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

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 of Python While Loop
Let's see a simple example of while loop in Python. The given Python code uses a 'while' loop to print "Hello Geek" three times by incrementing a variable called 'count' from 1 to 3.

```
count = 0
while (count < 3):
   count = count + 1
   print("Hello Geek")
```

Output
Hello Geek
Hello Geek
Hello Geek

For Loop in Python

For loops are used for sequential traversal. For example: traversing a list or string or array etc. In Python, there is "for in" loop which is similar to for each loop in other languages. Let us learn how to use for in loop for sequential traversals.

For Loop Syntax:

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

It can be used to iterate over a range and iterators.

The code uses a Python for loop that iterates over the values from 0 to 3 (not including 4), as specified by the range(0, n) construct. It will print the values of 'i' in each iteration of the loop.

```
n = 4
for i in range(0, n):
   print(i)
```

22 (A) Write a code in C to sort 10 names. 10 L3 3 5

```
#include<stdio.h>
#include<string.h>
main(){
  int i,j,n;
  char str[100][100],s[100];
```

```
  printf("Enter number of names :");
  scanf("%d",&n);
  printf("Enter names in any order:");
  for(i=0;i<n;i++){
    scanf("%s",str[i]);
  }
  for(i=0;i<n;i++){
    for(j=i+1;j<n;j++){
      if(strcmp(str[i],str[j])>0){
        strcpy(s,str[i]);
        strcpy(str[i],str[j]);
        strcpy(str[j],s);
      }
    }
  }
  printf("The sorted order of names are:");
  for(i=0;i<n;i++){
    printf("%s",str[i]);
  }
}
```

(OR)

(B) Discuss with examples the different methods used in a set. 10 L3 4 5

Set in Python is a collection of unique elements which are unordered and mutable. Python provides various functions to work with Set. In this article, we will see a list of all the functions provided by Python to deal with Sets.

Adding and Removing elements

We can add and remove elements form the set with the help of the below functions –

add(): Adds a given element to a set

clear(): Removes all elements from the set

discard(): Removes the element from the set

pop(): Returns and removes a random element from the set

remove(): Removes the element from the set