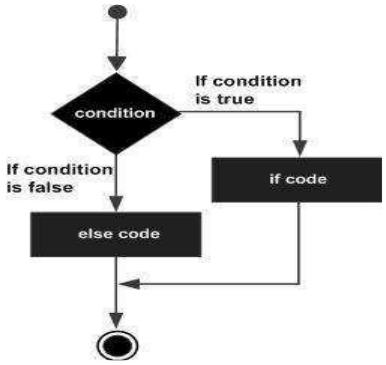
(Time: 2½ hours)

Total Marks: 75

N. B.: (1) **All** questions are **compulsory**.
(2) Make**suitable assumptions** wherever necessary and **state the assumptions** made.
(3) Answers to the **same question** must be **written together**.
(4) Numbers to the **right** indicate **marks**.
(5) Draw **neat labeled diagrams** wherever **necessary**.
(6) Use of **Non-programmable** calculators is **allowed**.

1.	**Attempt *any three* of the following:**	**15**
a.	**Write a note on features of python programming.** **Answer :** Features of python: Python's features include- • **Easy-to-learn:** Python has few keywords, simple structure, and a clearly defined syntax. This allows a student to pick up the language quickly. • **Easy-to-read:** Python code is more clearly defined and visible to the eyes. • **Easy-to-maintain:** Python's source code is fairly easy-to-maintain. • **A broad standard library:** Python's bulk of the library is very portable and crossplatform compatible on UNIX, Windows, and Macintosh. • **Interactive Mode:** Python has support for an interactive mode, which allows interactive testing and debugging of snippets of code. • **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms. • **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. • **Databases:** Python provides interfaces to all major commercial databases. • **GUI Programming:** Python supports GUI applications that can be created and ported 4 to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix. • **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. A few are listed below • It supports functional and structured programming methods as well as OOP. • It can be used as a scripting language or can be compiled to byte-code for building large applications. • It provides very high-level dynamic data types and supports dynamic type checking. • It supports automatic garbage collection. • It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.	5
b.	**What is variable? What are the rules and conventions for declaring a variable?** **Answer :** 1 mark for definition + 3 marks for rules + 1 mark for example	5

c.	**Explain the if—else statement with an example.**
	
	An **else** statement can be combined with an **if** statement. An **else** statement contains a block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value. The else statement is an optional statement and there could be at most only one **else** statement following **if**.
	Syntax
	The syntax of the **if...else** statement is-
	if expression:
	statement(s)
	else:
	statement(s)
	# Program checks if the number is positive or negative
	# And displays an appropriate message
	num = 3
	if num >= 0:
	print("Positive or Zero")
	
	 
	
		5

	else: print("Negative number") In the above example, when num is equal to 3, the test expression is true and body of if is executed and body of else is skipped.	

d.	**Write a python program to print factorial of a number. Take input from user.**
	Answer :
	# Python program to find the factorial of a number provided by the user.
	num = int(input("Enter a number: "))
	factorial = 1
	# check if the number is negative, positive or zero
	if num < 0:
	print("Sorry, factorial does not exist for negative numbers")
	elif num == 0:
	print("The factorial of 0 is 1")
	else:
	for i in range(1,num + 1):
	factorial = factorial*i
	print("The factorial of",num,"is",factorial)
		5

e.	**Explain continue statement with an example.**
	Answer :
	
	The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.
	
	Syntax of Continue
	
	Continue
	
	# Program to show the use of continue statement inside loops
	for val in "string":
	if val == "i":
	continue
	print(val)
	print("The end")
		5

f.	**Write a python program to calculate area of triangle and circle and print the result. Take input from user**.
	Answer :
	
	# Python Program to find the area of triangle
	a = float(input('Enter first side: '))
	b = float(input('Enter second side: '))
	c = float(input('Enter third side: '))
	# calculate the semi-perimeter
	s = (a + b + c) / 2
	# calculate the area
	area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
	print('The area of the triangle is %0.2f' %area)
	
	# Python Program to find the area of circle
	r = float(input('Enter radius: '))
	area=3.14* r*r
	print('The area of the circle is %0.2f' %area)
		5

2.	**Attempt *any three* of the following:**	**15**
a.	**Define function. Write syntax to define function. Give example of function definition**. **Answer :** 1 mark for definition + 2 marks for syntax + 2 marks for example	5

b.	**Write a python program to create a simple calculator using function that can add, subtract, multiply or divide depending upon the input from the user.**
	Answer:
	`# Program make a simple calculator that can add, subtract, multiply and divide using functions`
	
	`# This function adds two numbers`
	`def add(x, y):`
	`  return x + y`
	
	`# This function subtracts two numbers`
	`def subtract(x, y):`
	`  return x - y`
	
	`# This function multiplies two numbers`
	`def multiply(x, y):`
	`  return x * y`
	
	`# This function divides two numbers`
	`def divide(x, y):`
	`  return x / y`
	
	`print("Select operation.")`
	`print("1.Add")`
	`print("2.Subtract")`
	`print("3.Multiply")`
	`print("4.Divide")`
	
	`# Take input from the user`
	`choice = input("Enter choice(1/2/3/4):")`
	
	`num1 = int(input("Enter first number: "))`
	`num2 = int(input("Enter second number: "))`
	
	`if choice == '1':`
	`  print(num1,"+",num2,"=", add(num1,num2))`
	
	`elif choice == '2':`
	`  print(num1,"-",num2,"=", subtract(num1,num2))`
	
	`elif choice == '3':`
	`  print(num1,"*",num2,"=", multiply(num1,num2))`
	
	`elif choice == '4':`
	`  print(num1,"/",num2,"=", divide(num1,num2))`
	`else:`
	`  print("Invalid input")`
		5

c.	**Write a python program to calculate Fibonacci series using recursive function.**
	
	`def fibonacci(n):`
	`  if(n <= 1):`
	`  return n`
	`  else:`
	`  return(fibonacci(n-1) + fibonacci(n-2))`
	`n = int(input("Enter number of terms:"))`
	`print("Fibonacci sequence:")`
	`for i in range(n):`
	`  print fibonacci(i),`
		5

d.	**Discuss the local and global variable scope in python.**
	Answer:
	Global Variables
	In Python, a variable declared outside of the function or in global scope is known as global variable. This means, global variable can be accessed inside or outside of the function e.g. how a global variable is created in Python.
	
	x = "global"
	def foo():
	** print("x inside :", x)**
	foo()
	print("x outside:", x)
	
	When we run the code, the will output be:
	`x inside : global`
	`x outside: global`
	
	Create a Local Variable
		5

Normally, we declare a variable inside the function to create a local variable.

```
def foo():
    y = "local"
    print(y)
foo()
```

When we run the code, it will output:

```
local
```

Using Global and Local variables in same code

```
x = "global"
def foo():
    global x
    y = "local"
    x = x * 2
    print(x)
    print(y)

foo()
```

```
global global
local
```

e.	**Explain any five basic operations performed on string** **Answer :** 1 mark for each operation with an example	5

f	**Write a python program to check whether a string is Palindrome or Not**
	Answer :
	`str=input("Enter string:")`
	`if(str==str [::-1]):`
	`  print("The string is a palindrome")`
	`else:`
	`  print("The string isn't a palindrome")`
		5

3.	**Attempt *any three* of the following:**	**15**

a.	**What are lists? How to define and access the elements of list.**
	
	A list is an ordered set of values, where each value is identified by an index. The values that make up a list are called its elements. Lists are similar to strings, which are ordered sets of characters, except that the elements of a list can have any type. Lists and strings—and other things that behave like ordered sets—are called sequences.
	The list is the most versatile data type available in Python, which can be written as a list of comma-separated values (items) between square brackets.
	`# empty list`
	`my_list = []`
	`# list of integers`
	`my_list = [1, 2, 3]`
	`# list with mixed datatypes`
	`my_list = [1, "Hello", 3.4]`
	
	`# nested list`
	`my_list = ["mouse", [8, 4, 6], ['a']]`
	
	Values and Accessing Elements:
	
	The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.
	`my_list = ['p','r','o','b','e']`
	`# Output: p`
	`print(my_list[0])`
	`# Output: o`
	`print(my_list[2])`
		5

	# Output: e
	print(my_list[4])
	# Output: e
	print(my_list[-1])
	# Output: p
	print(my_list[-5])	

b.	**Write a program to input any two tuples and interchange the tuple values.**
	
	t1 = tuple( )
	t2=tuple()
	n = int(input('Total number of values in first tuple'))
	for i in range (n):
	  a = input('Enter elements')
	  t1 = t1 + (a,)
	
	n1 = int(input('Total number of values in second tuple'))
	for i in range (n1):
	  a = input('Enter elements')
	  t2 = t2 + (a,)
	
	print('Before Swapping :')
	print ('First tuple')
	print (t1)
	print ('Second tuple')
	print (t2)
	t1, t2 = t2, t1
	print ('After swapping')
	print ('First tuple')
	print(t1)
	print('Second tuple')
	print (t2)
		5

c.	Explain mkdir(), chdir() and rmdir methods in pthon. **The mkdir() Method** You can use the mkdir() method of the **os** module to create directories in the current directory. You need to supply an argument to this method, which contains the name of the directory to be created. **Syntax** os.mkdir("newdir") *import os* *# Create a directory "test"* *os.mkdir("test")* **The chdir() Method** You can use the chdir() method to change the current directory. The chdir() method takes an argument, which is the name of the directory that you want to make the current directory. **Syntax** *os.chdir("newdir")* *import os* *# Changing a directory to "/home/newdir"* *os.chdir("/home/newdir")* **The rmdir() Method** The rmdir() method deletes the directory, which is passed as an argument in the method. Before removing a directory, all the contents in it should be removed. **Syntax** os.rmdir('dirname')	5

import os
 # This would remove "/tmp/test" directory.
os.rmdir("/tmp/test")

d. How to create dictionary in python? Give example. 5

Answer :

Python dictionary is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair.

Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.

Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

```
# empty dictionary
my_dict = {}
# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}
# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}
# using dict()
my_dict = dict({1:'apple', 2:'ball'})
# from sequence having each item as a pair
my_dict = dict([(1,'apple'), (2,'ball')])
```

e. **Explain different modes of opening a file.** 5

Here is a list of the different modes of opening a file-

Modes	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading andwriting.
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

f. Write a python program to accept an integer number and use try/except to catch the exception If the input has not been a valid integer, generate a ValueError 5

Answer :

```
while True:
    try:
```

```
        n = input("Please enter an integer: ")
        n = int(n)
        break
    except ValueError:
        print("No valid integer! Please try again ...")
print "Great, you successfully entered an integer!"
```

4. **Attempt *any three* of the following:** **15**

a. **What is regular expression? What are different types of regular expression?** 5

Answer :

regular expression is a sequence of character(s) mainly used to find and replace patterns in a string or file. As I mentioned before, they are supported by most of the programming languages like python, perl, R, Java and many others. So, learning them helps in multiple ways (more on this later).
Regular expressions use two types of characters:
a) Meta characters: As the name suggests, these characters have a special meaning, similar to * in wild card.
b) Literals (like a,b,1,2…)
In Python, we have module "**re**" that helps with regular expressions. So you need to import library **re** before you can use regular expressions in Python.

import re

The most common uses of regular expressions are:

- Search a string (search and match.
- Finding a string (findall)
- Break string into a sub strings (split)
- replace part of a string (sub)

(2)Various types of regular expressions:

Basic patterns that match single chars

• **a, X, 9, <** -- ordinary characters just match themselves exactly.
• **. (a period)** -- matches any single character except newline '\n'
• **\w** -- matches a "word" character: a letter or digit or underbar [a-zA-Z0- 9_].
• **\W** -- matches any non-word character.
• **\b** -- boundary between word and non-word
• **\s** -- matches a single whitespace character -- space, newline, return, tab
• **\S** -- matches any non-whitespace character.
• **\t, \n, \r** -- tab, newline, return
• **\d** -- decimal digit [0-9]
• ^ = matches start of the string
• **$** = match the end of the string
• \ -- inhibit the "specialness" of a character.

Flag	**Meaning**
ASCII, A	Makes several escapes like \w, \b, \s and \d match only on ASCII characters with the respective property.
DOTALL, S	Make, match any character, including newlines
IGNORECASE, I	Do case-insensitive matches
LOCALE, L	Do a locale-aware match
MULTILINE, M	Multi-line matching, affecting ^ and $
VERBOSE, X (for 'extended')	Enable verbose REs, which can be organized more cleanly and understandably

b. **Explain math module with its any five functions.** 5

Answer : 1 mark for each function

c. **List and explain built in class attributes. Give example of use.** 5

Built-in Class Attributes:
Every Python class keeps the following built-in attributes and they can be accessed using dot operator like any other attribute −
• **dict :** Dictionary containing the class's namespace.
• **doc :** Class documentation string or none, if undefined.
• **name :** Class name.
• **module :** Module name in which the class is defined. This attribute is " main " in interactive mode.
• **bases :** A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

```
class Employee:
'Common base class for all employees' empCount = 0
def init (self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1

def displayCount(self):
       print ("Total Employee %d" % Employee.empCount)

def displayEmployee(self):
      print ("Name : ", self.name, ", Salary: ", self.salary)

emp1 = Employee("Zara", 2000) emp2 = Employee("Manni", 5000)
print ("Employee. doc :", Employee. doc )
print ("Employee. name :", Employee. name ) print ("Employee. module :", Employee. module ) print
("Employee. bases :", Employee. bases ) print ("Employee. dict :", Employee. dict )
```

d. **How to import a module? Explain time module.** 5

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.
Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.
You can use any Python source file as a module by executing an import statement in some other Python source file. The import has the following syntax-

```
import module1[, module2[,... moduleN]
```

When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module.

time Module :
There is a popular time module available in Python which provides functions for working with times and for converting between representations. Here is the list of all available methods −

Sr.No.	Function with Description
1	**time.altzone** The offset of the local DST timezone, in seconds west of UTC, if one is defined. This is negative if the local DST timezone is east of UTC (as in Western Europe, including the UK). Only use this if daylight is nonzero.
2	**time.asctime([tupletime])** Accepts a time-tuple and returns a readable 24-character string such as 'Tue Dec 11 18:07:14 2008'.
3	**time.clock( )** Returns the current CPU time as a floating-point number of seconds. To measure computational costs of different approaches, the value of time.clock is more useful than that of time.time().

	4	**time.ctime([secs])** Like asctime(localtime(secs)) and without arguments is like asctime( )		
	5	**time.gmtime([secs])** Accepts an instant expressed in seconds since the epoch and returns a time-tuple t with the UTC time. Note : t.tm_isdst is always 0		
	6	**time.localtime([secs])** Accepts an instant expressed in seconds since the epoch and returns a time-tuple t with the local time (t.tm_isdst is 0 or 1, depending on whether DST applies to instant secs by local rules).		
	7	**time.mktime(tupletime)** Accepts an instant expressed as a time-tuple in local time and returns a floating-point value with the instant expressed in seconds since the epoch.		
	8	**time.sleep(secs)** Suspends the calling thread for secs seconds.		
	9	**time.strftime(fmt[,tupletime])** Accepts an instant expressed as a time-tuple in local time and returns a string representing the instant as specified by string fmt.		
	10	**time.strptime(str,fmt='%a %b %d %H:%M:%S %Y')** Parses str according to format string fmt and returns the instant in time-tuple format.		
	11	**time.time( )** Returns the current time instant, a floating-point number of seconds since the epoch.		
	12	**time.tzset()** Resets the time conversion rules used by the library routines. The environment variable TZ specifies how this is done.		

e. **What is multithreading? How to create a thread?** 5

Answer:

Running several threads is similar to running several different programs concurrently, but with the following benefits −

- Multiple threads within a process share the same data space with the main thread and can therefore share information or communicate with each other more easily than if they were separate processes.
- Threads sometimes called light-weight processes and they do not require much memory overhead; they are cheaper than processes.

A thread has a beginning, an execution sequence, and a conclusion. It has an instruction pointer that keeps track of where within its context it is currently running.

- It can be pre-empted (interrupted)
- It can temporarily be put on hold (also known as sleeping) while other threads are running - this is called yielding.
-

Starting a New Thread

To spawn another thread, you need to call following method available in *thread* module −

```
thread.start_new_thread ( function, args[, kwargs] )
```

This method call enables a fast and efficient way to create new threads in both Linux and Windows.
The method call returns immediately and the child thread starts and calls function with the passed list of *args*. When function returns, the thread terminates.
Here, *args* is a tuple of arguments; use an empty tuple to call function without passing any arguments. *kwargs* is an optional dictionary of keyword arguments.

Example

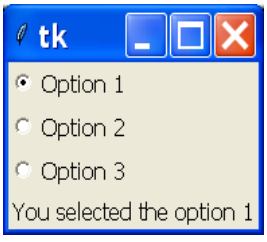
```
import thread
import time
# Define a function for the thread
def print_time( threadName, delay):
   count = 0
   while count < 5:
```

```
      time.sleep(delay)
      count += 1
      print "%s: %s" % ( threadName, time.ctime(time.time()) )
# Create two threads as follows
try:
   thread.start_new_thread( print_time, ("Thread-1", 2, ) )
   thread.start_new_thread( print_time, ("Thread-2", 4, ) )
except:
   print "Error: unable to start thread"
while 1:
   pass
```

f. **Design a class in python that store the information of student and display the same.** 5

```
class Student:

   'Common base class for all Students '

   studCount = 0


   def __init__(self, rollno, name):

      self.rollno =rollno

      self.name = name

      Student. studCount  += 1


   def displayCount(self):

     print "Total Student %d" % Student.studCount


   def display Student (self):

      print "Name : ", self.name,  ", Roll Number: ", self. rollno

"This would create first object of Student class"

stud1 = Student ("Zara", 2000)

"This would create second object of Student class"

stud2 = Student ("Manni", 5000)

stud1.display Student ()

stud2.display Student ()

print "Total Student %d" % Student.studCount
```

5.	**Attempt *any three* of the following:**	**15**
a.	**Write a Python code to create the following GUI:**	5




```
from tkinter import *
def sel():
    selection = "You selected the option " + str(var.get())
    label.config(text = selection)
    text.insert(END, selection)
root = Tk()
var = IntVar()
R1 = Radiobutton(root, text="Option 1", variable=var, value=1,command=sel)
R1.pack( anchor = W )
R2 = Radiobutton(root, text="Option 2", variable=var, value=2,command=sel)
```

```
R2.pack( anchor = W )
R3 = Radiobutton(root, text="Option 3", variable=var, value=3, command=sel)
R3.pack( anchor = W)
label = Label(root)
label.pack(anchor=W)
text = Text(root, width=30, height=1 )
text.pack(anchor=W)
root.mainloop()
```

b. **Explain the layout manager in detail.** 5

DIFFERENT LAYOUT MANAGERS IN PYTHON'S TKINTER GUI PACKAGE

- Pack Layout Manager explanation – 2 marks
- Grid Layout Manager explanation – 2 marks
- Place Layout Manager explanation – 1 mark

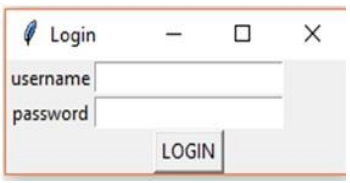
Explanation of each

c. **What is the use of listbox widget? Give an example to add elements to listbox.** 5

The **Listbox** widget is a standard Tkinter widget used to display a list of alternatives. The listbox can only contain text items, and all items must have the same font and color. Depending on the widget configuration, the user can choose one or more alternatives from the list.
Listboxes are used to select from a group of textual items. Depending on how the listbox is configured, the user can select one or many items from that list.

```
from Tkinter import *

master = Tk()

listbox = Listbox(master)
listbox.pack()
listbox.insert(END, "a list entry")

for item in ["one", "two", "three", "four"]:
    listbox.insert(END, item)

mainloop()
```

d. **Write a source code in python to create Login Screen.** 5




```
from tkinter import *

gui = Tk()
us1=StringVar()
pw1=StringVar()
def login():
    if us1.get() == 'Abc' and pw1.get()=='AAA':
        i=Label(gui,text='Login success').grid(row=6,column=0)
        print("login success")
    else:
        print("wrong password")
        j=Label(gui,text='Login failed').grid(row=6,column=0)
gui.title("Login")
a = Label(gui ,text="username").grid(row=0,column = 0)
b = Label(gui ,text="password").grid(row=1,column=0)
e = Entry(gui,textvariable=us1).grid(row=0,column=1)
f = Entry(gui,textvariable=pw1,show="*").grid(row=1,column=1)
##c = Button(gui, text="Create account").grid(row=2,column=0)
##a1 = Label(gui ,text="username").grid(row=3,column = 0)
##b1 = Label(gui ,text="password").grid(row=4,column=0)
##e1 = Entry(gui).grid(row=3,column=1)
##f1 = Entry(gui,show="*").grid(row=4,column=1)
c1 = Button(gui, text="LOGIN",command=login).grid(row=5,column=1)
gui.mainloop()
```

e. **Write a source code in python to read single and multiple results of query execution.** 5

```
import mysql.connector as mysql
conn = mysql.connect(user="root", password="", host='127.0.0.1',
```

```
db='myDB')
cursor = conn.cursor()
cursor.execute("SELECT * FROM employee")
row = cursor.fetchone()
print(row)
rows = cursor.fetchmany(2)
print(rows)
all = cursor.fetchall()
print(all)
conn.close()
```

f. **Write a source code in python to show database connectivity: Write a program to insert the following information:** 5

Table: Item

Item no	Item name	Price	Quantity
101	Geometry Box	50	100
102	Soap	100	50
103	Perfume	150	25
104	Pen	50	200
105	Pencil	20	100

Write queries based upon Item table given

1. **Display item name and price value.**
2. **Display the item information whose name starts with letter 'p'.**
3. **Display item name, whose price is in between 50 to 100.**

Answer :

Database connection &table creation code 1 mark
Record insertion code 1 mark
Each query 1 mark