S.Y. B.Sc. (IT) : Sem. III

Python Programming

Time : 2½ Hrs.] Prelim Question Paper Solution **[Marks : 75**

Q.1 Attempt the following (any THREE) **[15]**

Q.1 (a) Python programs are executed by an interpreter. There are two ways to use the interpreter. What are they? **[5]**

Ans.: Interpreter translates high level language into low level machine language. An interpreter reads the statement & first converts it into an intermediate code & executes it, before reading the next statement. It translates each instruction immediately one by one. There are two ways to use interpreter :

1. Interactive mode : It is used when we have few lines of code.
   Eg.
   ```
   >>>> 1+1
   2           /
   ```
   The prompt >>>> uses to indicate that it is ready
2. The script mode : To execute more than few lines of codes then interactive mode becomes clumsy so we can store code in a file & use the interpreter code in a file & use the interpreter to executes the contents of the file. Python scripts have .py ending. In script mode an expression by itself has no visible effect. It evaluates the expression & doesn't display the value enters we specify it to display.
   ```
   Eg.    miles = 26.2
          print (miles * 1.61)
          % 42.182
   ```

A script contains a sequence of statements. If there are more than one statement the result appears one at a time.

```
print (1)
x = 2
print (x)
o/p
1
2
```

To execute the script the name of the file is provided.

Q.1 (b) A triple of numbers (a,b,c) is called a Pythagorean triple if a * a + b * b = c * c. In this question, you will be given three numbers. You have to output 1 if the three numbers form a Pythagorean triple. Otherwise, you have to output 0.
Note that the inputs may not be given in order: you may have to try all possible orderings of the three numbers to determine whether they form a Pythagorean triple. **[5]**

Ans.:
```
def pytha();
a = int (input ("a=  "))
b = int (input ("b=  "))
c = int (input ("c=  "))
if ((a * a + b * b = = c * c = = a * a));
d = 1
else :
d = 0
print (d)
o/p
>>> python ()
a = 3
b = 5
c = 4
1
```

Q.1 (c) Explain type conversion of variable in Python. [5]

Ans.: **Type conversion :** To convert between types you simply use the type name as a function. In addition, several built in functions are supplied to perform special kinds of conversions. All of these functions returns a new object representing the converted value.

(i) int(x) – converts x to a plain integer
Example : int(45.67)   Output : 45

(ii) long(x) – converts x to a long integer
Example : long (67.898)   Output : 67

(iii) float(x) – converts x to a floating point number.
Example : float (34)   Output : 34.0

(iv) Str(x) – converts object X to a string representation.
Example : Str(2)   Output : '2'

(v) Unichr (x) – converts an integer to a Unicode character.
Example : Unichar(65)   Output : A

(vi) Complex (real[img]) : Creates a complex number.
Example : Complex(45, 4)   Output : (45+4j)

Q.1 (d) Explain if….else statement with example. [5]

Ans.: **Syntax :**

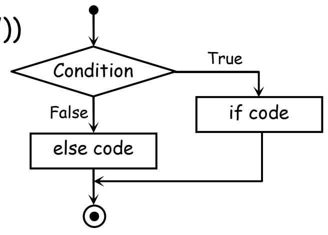
```
If test – expression:
    statements
else;
    staatements
```

The if … else statement evaluates the test expression and will execute body of if only when condition is true otherwise else is executed.

Example :

```
    temp = float (input ("what is the temperature"))
    iftemp > 48:
        print ("It is too hot")
    else :
        print("It is not too hot")
```

Flow diagrams:



Q.1 (e) What is the difference between interactive mode and script mode in Python? [5]

Ans.: **Python has 2 basic modes :** Script and interactive. The normal mode is the mode where the scripted and finished .py files are run in the Python interpreter. Interactive mode is a command like shell which gives immediate feedback for each statement, while running previously fed statements in active memory.

Interactive Mode : In this mode Python displays the result of expressions. It allows interactive testing and debugging of snippets of code.

```
Example: >>> print(5×7)
               35
```

Script Mode : In a script mode code is written in a separate file and it is saved with an extension .py. Then it can be executed by selecting 'Run module' option from Run Menu. (Shortcut F5)

Example : File name is \.py

```
a = 10
if (a > 5);
```

```
    print("Greater")
else :
    Print("Smaller")
```

Output :

```
Greater
```

Q.1 (f) Explain the use of break statement in a loop with example. [5]

Ans.: The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.

If break statement is inside a nested loop, breaks will terminate the innermost loop.

Syntax :

```
    break
```

Example : Program to find out whether the input number is prime or not.

```
    n = int(input("Enter a no."))
    for i in range(2, n+1):
    if n% i ==0:
            break
    if i == n:
            print(n, 'is prime')
    else:
            print(n, 'is not prime')
```

Output :

```
1st Run
Enter a no. 5
5 is prime
2nd Run
Enter a no 10
10 is not prime
```

Q.2 Attempt the following (any THREE) [15]

Q.2 (a) (i) Name the following code and explain the same: [5]

```
def recurse():
        recurse()
```

Ans.: def recurse(): Recursive function defined
  recurse() call to function

A function that calls itself is known as a recursive function & this technique is known as recursion.

Eg.
```
def fact (n):
        if n = = 1:
            return n
        else :
            return n*fact (n–1)
num = int (input ("Enter a number:"
if num > 0:
    print ("factorial doesn't exist")
else :
    print ("Factorial is" + str (num) + "is + str (fact (num)))
```

Q.2(a) (ii) **[5]**

```
def countdown(n):
        if n <= 0:
                print ('Blastoff!')
        else:
        print( n)
        countdown(n-1)
```

What will happen if we call this function as seen below?
>>> countdown(3)

Ans.:

```
def countdown (n) :
    if n < = 0:
        print ("Blastoff")
        else :
        print (n)
        countdown (n–1)
>>> countdown(3)
    o/p
    3
    2
    1
    Blastoff!
```

Q.2 (a) (iii) **Fermat's Last Theorem says that there are no positive integers a, b, and c such that $a^n + b^n = c^n$ for any values of n greater than 2.** **[5]**
Write a function named check_fermat that takes four parameters—a, b, c and n—and that checks to see if Fermat's theorem holds. If n is greater than 2 and it turns out to be true that $a^n + b^n = c^n$ the program should print, "Holy smokes, Fermat was wrong!" Otherwise the program should print, "Fermat's Theorem holds."
Write a function that prompts the user to input values for a, b, c and n, converts them to integers, and uses check_fermat to check whether they violate Fermat's theorem

Ans.:

```
def check_fermat (a, b, c, n) :
    if n > 2 :
        if (a ** n + b ** n == c**n :
            print ("Holy smokes, fermat was wrong!")
        else :
            print ("a ** {0} + b ** {0} == c** {0} is false" . format (n))
        elif (a**n + b**n) == c**n :
            print ("fermat's theorem holds")
        else :
            print ("a**{0} + b**{0} == c**{0} is false" .fermat(n))
def f() :
    a = int (input ('a = '))
    b = int (input ('b = '))
    c = int (input ('c = '))
    n = int (input ('n = '))
    check_fermat (a, b, c, n)
```

Q.2 (b) (i) Write the output for the following if the variable fruit = 'banana' : [5]

```
>>> fruit[:3]
>>> fruit[3:]
>>> fruit[3:3]
>>> fruit[:]
```

Ans.:

```
fruit = 'banana' :
>>> fruit [:3]
   'ban'
>>> fruit [3:]
   'ana'
>>> fruit [3:3]
   ' '
>>> fruit [:]
   'banana'
```

Q.2 (b) (ii) What is the reason for the error seen here? [5]

```
>>> greeting = 'Hello, world!'
>>> greeting[0] = 'J'
TypeError: 'str' object does not support item assignment
```

Ans.:

```
>>> greeting = 'Hellow.world!'
>>> greeting [0] = 'J'
```

It will show error because strings or immutable. We cannot change an existing string.

Q.2 (b) (iii) Explain any 2 methods under strings in Python with suitable examples.

Ans.: (a) Concatenation operation is done with the + operator in Python. Concatenation means joining the string by linking the last end of the first string with the first end of the second string. Two separate strings transform into the one single string after concatenation.

```
>>> "Hello" + test
'Hellotest'        o/p
```

(b) Repetition operation is performed on the strings in order to repeat the stings several times. It is done with * operator.

```
>>> 'Hello' * 3
    HelloHelloHello    o/p
```

Q.2(c) Short note on incremental development. [5]

Ans.: If the code is bigger we have to spend the maximum time to debug. To simply this task of dealing with large code we can use process called incremental development. The goal of incremental development is to avoid long debugging session by adding and testing only small amount of code at a time.

The key aspects of the process are :

1. Start with a working program and make small incremental changes. If any error you will know exactly where it is.
2. Use temporary variables to hold intermediate values so you can output and check them.
3. Once the program is working, you might want to remove some of scaffolding or consolidate multiple starts into compound expressions but only if it does not make program difficult to read.

Example : Find area of a circle, given by radius r.
The formula is Area = πr^2

Step 1: Establish the input output. In this case the r is the input which can represent using one parameter. The return value is the area, which is floating point value.

Step 2: Build the code block that calculates the area of a circle.

```
>>> defcarea(r) :
        Output 0.0
        return 0.0
```

It doesn't compute. It always return 0 but it is syntactically correct and we context syntax before we make it more complicated.

Step 3: Now apply formula (3.14*r*r)

```
>>> def carea(r)
        return 3.14*r*r
```

Q.2(d) Explain str.find( ) function with suitable example. **[5]**

Ans.: str.find() function determines if string str occurs in string or in a substring of string if starting index beg and ending index end are given :

Syntax : str.find(str,beg=0, end=len(string))

Parameters:

- str – This specifies the string to be searched.
- beg – This is the starting index, by default its 0.
- end – This is the ending index, by default its equal to length of the string.

Return value
Index if found and –1 otherwise

Example :

```
str1 = "This is University of Mumbai";
str2 = "Uni"
print(str1.find(str2))
print(str1.find(str2, 10))
print(str1.find(str2, 40))
```

Output:

```
8
–1
–1
```

Q.2(e) Write a Python code to check whether the given number is a strong number. Strong Numbers are numbers whose sum of factorial of digits is equal to the original number (Example: 145 = 1! +4! + 5!) . **[5]**

Ans.:

```
def strongno (n) :
    n1 = n
    a = 0
    while n! = 0 :
        d = n% 10
        i = 1
        f = 1
        while i < = d :
            f = f * i
            i = i + 1
        s = s + f
        n = n// 10
    if s == n! :
        print ("{0} is a strong No" . format(n1))
    else :
        print ("{0} is not a strong No" . format (n1))

output
>>> strongno (145)
    145 is a strong No
```

Q.2(f) The Ackermann function, A(m,n), is defined: **[5]**

$$A(m,\ n) = \begin{cases} n+1 & \text{if } m=0 \\ A(m-1,1) & \text{if } m>0 \text{ and } n=0 \\ A(m-1, A(m, n-1)) & \text{if } m>0 \text{ and } n>0 \end{cases}$$

Write a recursive function named ackermann that evaluates Ackermann's function.

Ans.:

```
def Ackermann (m, n) :
    if m = = 0 :
        return n + 1
    if n = = 0 :
        return ackermann (m – 1, 1)
    return ackermann (m – 1, ackermann (m, n – 1))
m = int (input ("m = "))
n = int (input ("n = "))
print ("Ackermann ({0}, {1} = {2}" . format (m, n ackermann (3, 4)))
```

Q.3 Attempt the following (any THREE) **[15]**

Q.3 (a) (i) Write the output for the following : **[5]**

```
(1) >>> a = [1, 2, 3]
    >>> b = [4, 5, 6]
    >>> c = a + b
    >>> print c
(2) >>> [1, 2, 3] * 3
(3) >>> t = ['a', 'b', 'c', 'd', 'e', 'f']
    >>> t[1:3] = ['x', 'y']
    >>> print t
```

(ii) Explain any 2 methods under lists in Python with examples.

(iii) Write the function tail(l) to get the following output:

```
>>> letters = ['a', 'b', 'c']
>>> rest = tail(letters)
>>> print rest
    ['b', 'c']
```

Ans.:

```
(1) >>> a = [1, 2, 3]
    (i) >>> b = [4, 5, 6]
    >>> c = a + b
    >>> print (c)
    [1, 2, 3, 4, 5, 6]

2.  >>> [1, 2, 3] * 3
    >>> [1, 2, 3] * 3
    [1, 2, 3, 1, 2, 3, 1, 2, 3]

3.  >>> t = ['a', 'b', 'c', 'd', 'e', 'f']
    >>> t[1:3] = ['x', 'y']
    >>> print t

    >>> t = ['a', 'b', 'c', 'd', 'e', 'f']
```

(ii) Methods under lists

1. Append : It adds a new element to the end of a list.
```
>>> t = ['a', 'b', 'c']
>>> t . append ('d')
>>> print t
o/p → ['a', 'b', 'c', 'd']
```

2. extend – it takes list as an argument and appends all of the elements
```
>>> t1 = ['a', 'b', 'c']
>>> t2 = ['d', 'e']
>>> t1 . extend (t2)
>>> print t1
o/p → ['a', 'b', 'c', 'd', 'e']
```

(iii)
```
def tail (l)
return l [1 :]
letters = ['a', 'b', 'c']
rest = tail (letters)
print (rest)

o/p >>> ['b', 'c']
```

Q.3 (b)List and explain any five exceptions in Python. **[5]**

Ans.: **Exceptions :** The runtime error are also called as exceptions as they usually indicate that something exceptional has happened.

These are most common suntime errors are:

(i) Name Error: Trying to use a variable that doesn't exist in the current environment.

(ii) Type Error: Trying to use a value improperly or a mismatch between items in a format string and items paned or paning wrong number of arguments.

(iii) Key Error: Trying to access an element of a dictionary using a key that the dictionary does not contain.

(iv) Attribute error: Trying to access an attribute or method that does not exist.

(v) Index Error: The index we are using to access a list, string as triple in greater than its length mins one.

Q.3 (c) Explain the following statements: **[5]**

(i) >>> a, b = 1, 2, 3 **[5]**

(ii) >>> addr = 'monty@python.org'
>>> uname, domain = addr.split('@')

(iii) >>> t = divmod(7, 3) **[5]**
>>> print(t)

(iv) >>> s = 'abc' **[5]**
>>> t = [0, 1, 2]
>>> zip(s, t)

(v) for index, element in enumerate('abc'): **[5]**
print(index, element)

Ans.: (i)
```
>>> a, b = 1, 2, 3
```
Valve error: two many valves to unpack

(ii)
```
>>> addr = 'manty@python.org'
>>> uname, domain = addr.split ('@')
```
It splits in two parts and uname gets monty demain gets python.org

(iii)
```
t = divmod (7, 3)
>>> print(t)
```
Output (2, 1)

(iv) >>> s = 'abc'
>>> t = [0, 1, 2]
>>> zip (s, t)
Output {('c',2), ('a', 0), ('b', 1)}

(v) for index, element in enumerate ('abc')
print (index, element)

```
output      0   a
            1   b
            2   c
```

Q.3 (d) What is tuple in Python? How to create and access it? [5]

Ans.: A tuple is a sequence of immutable Python objects. Tuples are sequences, just like list. The differences between tuples and lists are, the tuples cannot be changed unlike list and tuples use parentheses whereas list use square brackets.
To creat tuple use () parentheses and values must be comma– separated.
Eg. tup1 = ('physics', chemistry', 1900, 2000)
tup2 = (1, 2, 3, 4, 5)
tup3 = "a", "b", "c"
like string indices, tuple indices start at 0, and they can be sliced, concentrated and so on.

To access values in tuple:
To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example

```
tup1 = ('Phy', 'che', 1900, 2000)
tup2 = (1, 2,3,4, 5, 6, 7)
print "tup1[0]:", tup1[0]
print "tup2[1:5]:", tup [1:5]
```

Output :

```
tup1[0]: phy
tup2[1:5]: [2, 3, 4, 5]
```

Q.3 (e) Explain open ( ) and close ( ) methods for opening and closing file. [5]

Ans.: Python has built in function open () to open a file. This function returns a file object also called a handle as it is used to read or modify the file accordingly.
Syntax :
obj = open ("path of file" [mode])
The default is reading in text mode.
Following are the file modes

Mode	Description
r	Open a file for reading (default)
w	Open a file for writing.
x	Open a file for exclusive creation.
a	Open for appending at the end of file
b	Open in binary mode
t	Open in text mode (default)
+	Open a file for updating (read and write)

Example : f = Open("text.txt", \w) # write intext mode

close() file
when we are done with operations to the file, we need to properly close the file. Closing a file will free up the resources that were tied with the file and is done using Python closes method.

```
f = open ("test.txt")
# perform file operations
f.close()
```

Q.3 (f)Write short note on Tuple Assignment (Packing and Unpacking) [5]

Ans.: **Tuple Assignment:** A tuple of variables on the left of an assignment can be assigned values from a tuple or the right of the assignment.

```
T = ('Tushar', 34, 'M', 20000, 'Mumbai')
(name, age, gender, income, place) = T
```

The above statements do assignment of single tuple T to the all seven variables in other tuple, which saves seven assignment statements. The tuple assignment can be thought as tuple packing/unpacking.

In packing the values on the left are 'packed' together in a tuple:

```
>>> a = ('Tush', 34, 'IT')       # tuple packing
```

In unpacking, the values in a tuple on the right are 'unpacked' into the variables/names on the right:

```
>>> b = ('Sonu', 32, 'CS')
>>> (name, age, studies) = b  # tuple unpacking
>>> name
'Sonu'
>>> age
32
>>> studies
'CS'
```

The left side is a tuple of variables; the right side is a tuple of values. Each value is assigned to its respective variable. All the expressions on the right side are evaluated before any of the assignments. This is one the best feature of tuple. But remember here that the number of variables on the left and the number of values on the right have to be the same. IF not it will raise error as shown below.

```
>>> (a, b, c, d) = (1, 2, 3)
ValueError: need more than 3 values to unpack
```

Q.4 Attempt the following (any THREE) [15]

Q.4 (a) Explain the role of the Regular Expressions in the following snippets: [5]

(i)
```
>>> p = re.compile('\d+')
>>> p.findall('12 drummers drumming, 11 pipers piping, 10 lords a-leaping')
```
(ii)
```
>>> p = re.compile('ca+t')
>>> p.findall('ct cat caaat caaaaaat caaat ctttt cccttt')
```
(iii)
```
>>> p = re.compile('ca*t')
>>> p.findall('ct cat caaat caaaaaat caaat ctttt cccttt')
```
(iv)
```
>>> p = re.compile('a/{1,3}b')
>>> p.findall('a/b a//b a///b a/////b ab')
```
(v)
```
>>> result=re.findall(r'\w*','AV is largest Analytics community of India')
>>> print (result)
```

Ans.: (i)
```
>>> p = re.compile('\d+')
>>> p.findall('12 drummers drumming, 11 pipers piping, 10 lords a-leaping')
Output ('12', '11', '10')
```

(ii)
```
>>> p = re.compile('ca+t')
>>> p.findall('ct cat caaat caaaaaat caaat ctttt cccttt')
Output ['cat', 'caat', 'caaaaaat', 'caat']
```

(iii)
```
>>> p = re.compile('ca*t')
>>> p.findall('ct cat caaat caaaaaat caaat ctttt cccttt')
```

```
     Output ['ct', 'caaat', 'caaaaaat', 'caaat']
(iv) >>> p = re.compile('a/{1,3}b')
     >>> p.findall('a/b a//b a///b a/////b ab')
     Output ['a/b', 'a//b', 'a///b']

(v)  >>> result=re.findall(r'\w*','AV is largest Analytics community of India')
     >>> print (result)
     ['AV', '', 'is', ' ', 'largest', ' ', 'Analytics', ' ', 'community', '  ', 'of', '  ', 'India', ' ' ]
```

Q.4 (b) Explain Class inheritance in Python with an example. **[5]**

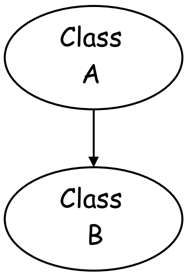
Ans.: Class inheritance : The mechanism of designing or constructing classes from other classes in called inheritance.

Syntax

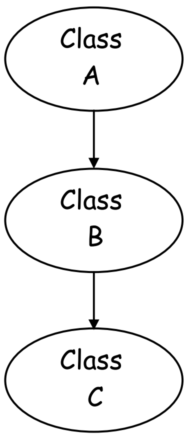
```
"class BaseclassName,
    Body of base class
class derived Class Name (BaseClassName);
    Body of derived class
```

Types of inheritance

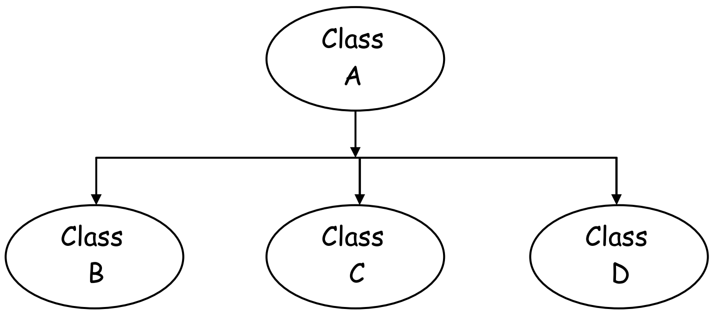
Single inheritance



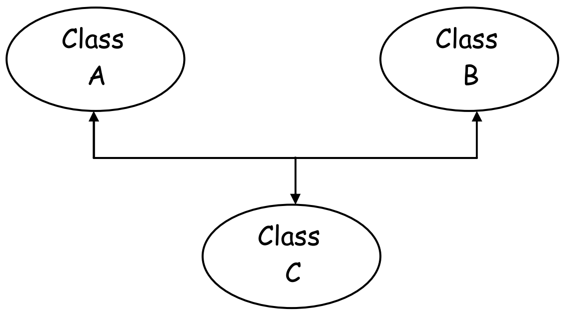
Multilevel inheritance



(iii) Hierarchical inheritance



(iv) Multiple inheritance



Example: Single inheritance

```
# super class
   Class person:
     def      _init_(self, name, age);
              self.name = name
              elf.age = age
     def      get_info(self);
              return(self.name + " " + str(self.org))
     def      is employee (self);
              returns false
     class    Employee (person);
     def      is Employee(salf);
              return true
     emp = person("xyz", 10)
     point (emp.getinfo( ). Emp is employee( ))

     emp = Employee ("PQR", 20)
     print (emp.getinfo( ), emp.is Employee( ))
```

```
output
    xyz    10   false
    PQR    20   true
```

Q.4 (c) (i) How is method overriding implemented in Python? [5]
(ii) Which methods of Python are used to determine the type of instance and inheritance?

Ans.: (i) **Method overriding** : It is the ability to define the method with the same name out different number of arguments and data types.
In python we define method without the types of parameters. So method overloading is implicitly class method overriding. If the parent and child class have the same method and if the method is called by the child class object then the child class verision is called overiding the parent class.
This is called method overiding.

```
Class parent (object):
    def_init_(self):
        self.valve = 5
    def get_valve(self):
        returns salf.valve
class child (Parent):
    def_get_valve(self):
        return self.valve + 1
c = child( )
print (c.get_valve( ))
```

Output 6

Here parent class has get_valve(self) method and the child class also have the same method.

C is a abject of child class which calls the Derived class version of the method overiding the parent class.

(ii) isinstance( ) it returns true if the object is an instance of the class or other classes derived from it.
Isinstance( ) also checks whether a class in a subction or not.
Every class in python inherits from the bare class object.

Q.4 (d) Explain match ( ) function with suitable example. [5]

Ans.: **Match function** – This function attempts to match RE pattern to string with optional flags.
Syntax – re.match (pattern, string, flags = 0) where pattern is the regular expression to be matched.
String – It is the string which would be searched to match the pattern at the beginning of string.
Flags – you can specify different flags using bitwise OR (1)

Match object methods & description :

- group (num = 0) – This method returns entire matter.
- groups () – This method return all matching subgroups in a tuple.

Example :

```
import re
line = "cats are smarter than dogs"
obj = re. match (r % *) are (. * ?) *', line /re. M\re.l)
if obj:
print "matchobj.group (1) :", obj.group (1)
print "matchobj.group (2) :", obj.group (2)
else:
print "No match"
```

Output:

```
match obj.group (1) : cats
match obj.group (2) : smarter
```

Q.4 (e) What is module? What are the advantages of using module? [5]

Ans.: Module – 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.

Advantages : Python provides the following advantages for using module:

(i) Reusability : Module can be used in some other python code. Hence it provides the facility of code reusability.
(ii) Categorization : Similar type of attributes can be placed in one module.
(iii) Python modules are easy to write and to import.

Q.4 (f) Explain various functions of math module. [5]

Ans.: (i) Ceil (x) – returns the smaller integer greater than or equal to x.
(ii) fabs (x) – returns the absolute value of x.

(iii) factorial (x) – returns the factorial of x.
(iv) floor (x) – returns the largest integer less than or equal to x.
(v) sart (x) – returns the square root of x.

```
import math :
print ("flour output =", math.floor (1000.5))
print ("ceil output =", math.ceil (1000.5))
print ("trunc output =", math.trunc (123.45))
print ("power output =", math.pow (2,3))
print ("square root output =", math.sart (9))
print ("absolute output = ", math.fabs (–123.5))
```

Output :

```
floor output = 1000
ceil output = 1001
trunc output = 123
power output = 8.0
square root output = 3.0
absolute output = 123.5
```

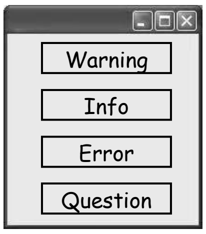
Q.5 Attempt the following (any THREE) **[15]**

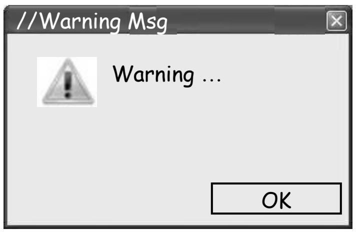
Q.5 (a) Write a Python code to show the following types of message boxes: **[5]**

Ans.:

```
from tkinter import *
root = TK( )
def m1( ) :
    messagebox.show warning("Warning Msg", "Warning")
def m2( ) :
    messagebox.show info("info msg", (Good Morning")
def m3( ) :
    messagebox, showerror("Error msg", "Error")
def m4( ) :
    messagebox, askquestion("Question msg", "Do u want to continue?")
Button(root, text = "warning", command = m1).pack( )
Button(root, text = "Info", command = m2).pack( )
Button(root, text = "Error", command = m3).pack( )
Button(root, text + "Question", command = m4).pack( )
root.mainloop( )
```

Output :




//Warning Msg

Warning ...

Q.5 (b) Write a Python code to do the following: [5]
(i) Create a table EMPLOYEE with columns Id (integer, not null, primary key), AdhaarNo (integer, not null), Empname (not null, varchar), Empdept (not null, varchar) and Empage (not null, integer) in MySQL and insert 5 rows of data.
(ii) Update the respective rows to Empdept = 'ERP' where Empage is >=40.

Ans.:

```
import mysql.connection as ms
con = ms.connect(uses = "root", panword = "root", host = "localhost", database = "students")
cur = con.cursor( )
cur_execute ("create table emp2 (Id int not null auto_increment,
                AdhaarNo int not null, Empname varchar(25) not null,
                Empdept varchar(15) nut null, Empage int not null,
                primary key(id)")
cur.execute ("Insert into emp2 (AdhaarNo, Empname, Empdept, Empage)
            values (700, 'Geeta", VSIT, 34)")
cur.execute("Insert int emp2 (AdhaarNo, Empname, Empdept, EMpage)
            values(900, "Smita', VSIT, 37)
con.commit( )
cur.execute ("select * from emp2")
for(Id, AdhaarNo, EmpName, Empdept, Empage) in cur :
    print ("Id : ( ), AdhaarNo : ( ), Name : ( ), Dept : ( ), Age : ( ) \n"
    format (Id, adhaarNo, Empname, Empdept, Empage))
cur.execute ("update emp2 set empdept = "'ERP' where Empage >= 36")
cur.execute("select * from emp2")
for (Id, AdhaarNo, EmpName, Empdept, Empage) in cur :
print ("Id : ( ), AdhaarNo : ( ), Name : ( ), Dept : ( ), Age : ( ) \n".
    format (Id, AdhaarNo, Empname, Empdept, Empage))
con.close( )
```

Output :

```
Id : 1, Adhaar No.: 700, Name : Geeta, Dept : VSIT, Age : 34
Id : 2, Adhaar No.: 800, Name : Babita, Dept : VSIT, Age : 35
Id : 3, Adhaar No.: 900, Name : Sunita, Dept : VSIT, Age : 37

Id : 1, ......          ......          ......          ...
Id : 2, ......          ......          ......          ...
Id : 3, AdhaarNo. : 900, Name : Sunita, Dept : ERP, Age : 37
```

Q.5 (c) Explain Checkbutton widget with example. [5]

Ans.: The checkbutton widget is standard Tkinter widgets used to implement on–off selections. It can contain text or images and you can associate a python function or method with each button. When the button is pressed, Tkinter calls that function or method.

Syntax :

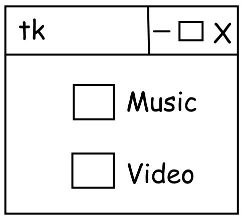
```
W = Checkbutton (master, option, ...)
```

Where,
Master – This represents the parent window.
Option – Here is the list of most commonly used options for this widget. These options can be used as key – value pairs separated by commas.

Example:

```
From Tkinter import *
Import theMessageBox
import Tkinter
```

```
Top = Tkinter . Tk ()
VC1 = Intvar ()
VC2 = Intvar ()
C1 = Checkbutton (top, text = "Music", variable = VC1, onvalue = 1,
offvalue = 0, height = 5, width = 20)
C2 = Checkbutton (top, text = "Video", variable = VC2, onvalue = 1,
offvalue = 0, height = 5, width = 20)
C1 . pack ()
C2 . pack ()
```



Q.5 (d) Write short note on cursor object in Python. **[5]**

Ans.: Cursor object represent a db cursor, which is used to manage the context of a fetch operation. Cursors created from the same connection are not isolated that is any changes done to the db by a cursor are immediately visible by the other cursors.

Cursors are created by the connection. Cursor () method they are bound to the connection for the entire lifetime and all the commands are executed in the context of the db session wrapped by the connection.

Methods

- open () – opens the cursor.
- close () – closes the cursor.
- fetchone () – This method retrieves the next row of a query result set and returns single sea or home if no more rows are available.

eg.1
```
cur = con . cursor()
cur . execute ("select * from employee")
cur . close()
```

eg.2
```
cur . execute ("select * from employee")
row = cursor.fetchone()
while row is not none :
print (row)
row = cursor.fetchone ()
```

eg.3
```
cursor.execute ("select * form employee")
for row in cursor :
print (row)
```

Q.5 (e) What is layout management? Explain Grid manager. **[5]**

Ans.: Layout Management is the act of arranging the widgets in the window at proper place for better display and easy to access for the user. When we design the GUI of our application, we decide what widgets we will use and how we will organize those widgets in the application. To organize our widgets we use specialised non–visible objects called layout managers.

Pack Manager – The pack geometry manager packs widgets in rows or columns. You can use options like fill, expand and side to control this.

Syntax :

    widget.pack (pack_options)

where

- expand – when set to true, widget expands to fill any space not otherwise used in widgets parent.
- fill – Determines whether widget fills any extra space allocated to it by the packer, or keeps its own minimal dimensions : None, X (horizontal), V(vertical), or both (horizontal & vertical)
- Side – Determines which side of the parent widget packs against : Top (default, bottom, left or right.

  Example : b1 = Button (frame, text = "red")
  b1.pack (side = LEFT)

Q.5 (f)Short note on Radio button creation. [5]

Ans.: A radio button, sometimes called option button, is a graphical user interface element of Tkinter, which allows the user to choose (exactly) one of a predefined set of options. Radio buttons can contain text or images. The button can only display text in a single font. A Python function or method can be associated with a radio button. This function or method will be called, if you press this radio button.

Radio buttons are named after the physical buttons used on old radios to select wave bands or preset radio stations. If such a button was pressed, other buttons would pop out, leaving the pressed button the only pushed in button.

Each group of Radio button widgets has to be associated with the same variable. Pushing a button changes the value of this variable to a predefined certain value.

Simple Example With Radio Buttons

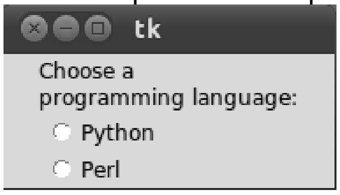
```
import tkinter as tk

root = tk.Tk ( )

v = tk.IntVar( )

tk.Label (root,
             text = " " "Choose a
programming language:" " ",
          justify = tk.LEET,
          padx = 20) . pack( )
tk.Radiobutton (root,
          text = "Python",
          padx = 20,
          variable = v,
          value = 1) · pack (anchor = tk.w)
tk.Radiobutton (root,
          text = "Perl",
          padx = 20,
          variable = v,
          value = 2) . pack (anchor=tk.w)
root.mainloop( )
```

The result of the previous example looks like this:



❑ ❑ ❑ ❑ ❑