VIDYALANKAR SCHOOL OF INFORMATION TECHNOLOGY

S.Y.B.Sc.I.T. – SEM - III Subject: Python Programming

UNIT 1: Unit name

Q1 Python programs are executed by an interpreter. There are two ways to use the interpreter. What are they?

Ans:

Interpreter translates high level language into low level machine language. An interpreter reads the statement and first converts it into an intermediate code and executes it, before reading the next statement. It translates each instruction immediately one by one.
There are two ways to use the interpreter: interactive mode and script mode.
1) In interactive mode—It is used when we have few lines of code. Type the Python programs and the interpreter displays the result:

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

The chevron, >>>, is the prompt the interpreter uses to indicate that it is ready. If we type 1 + 1, the interpreter replies 2.
2) The script mode—If we have to execute more than few lines of code then interactive mode becomes clumsy so we can store code in a file and use the interpreter to execute the contents of the file, which is called a script. By convention, Python scripts have names that end with .py
In script mode an expression , by itself has no visible effect. It evaluates the expression and doesn't display the value unless we specify it to display(in contrast to interactive mode where output of expression is shown immediately).
Eg miles=26.2

```
print(miles*1.61)
```

A script contains a sequence of statements. If there are more than one statement , the results appear one at a time.

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

output will be

```
1
2
```

To execute the script, we have to tell the interpreter the name of the file.

Q2 Explain **for loop** in Python with an example code to print Fibonacci series up to 10 terms.

Ans: A lot of computations involve processing a string one character at a time. Often they start at the beginning, select each character in turn, do something to it, and continue until the end. This pattern of processing is called a **traversal.**

```
for char in fruit:
print char
```

Each time through the loop, the next character in the string is assigned to the variable char.The loop continues until no characters are left.
Example

```
prefixes = 'JKLMNOPQ'
suffix = 'ack'
for letter in prefixes:
```

```
print letter + suffix
```

The output is:

```
Jack
Kack
Lack
Mack
Nack
Oack
Pack
Qack
```

Fibonacci series

```
Number = int(input("\nPlease Enter the Range Number: "))

# Initializing First and Second Values of a Series
First_Value = 0
Second_Value = 1

# Find & Displaying Fibonacci series
for Num in range(0, Number):
        if(Num <= 1):
                Next = Num
        else:
                Next = First_Value + Second_Value
                First_Value = Second_Value
                Second_Value = Next
        print(Next)
```

Output

```
Please Enter the Range Number: 6
0
1
1
2
3
5
```

Q3 Ans: Explain features of Python programming language.

Python Features

There are a lot of features provided by python programming language.

1) Easy to Use:

Python is easy to very easy to use and high level language. Thus it is programmer-friendly language.

2) Expressive Language:

Python language is more expressive. The sense of expressive is the code is easily understandable.

3) Interpreted Language:

Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.

4) Cross-platform language:

Python can run equally on different platforms such as Windows, Linux, Unix , Macintosh

etc. Thus, Python is a portable language.
5) Free and Open Source:
Python language is freely available(www.python.org).The source-code is also available. Therefore it is open source.
6) Object-Oriented language:
Python supports object oriented language. Concept of classes and objects comes into existence.
7) Extensible:
It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in your python code.
8) Large Standard Library:
Python has a large and broad library.
9) GUI Programming:
Graphical user interfaces can be developed using Python.
10) Integrated:
It can be easily integrated with languages like C, C++, JAVA etc.

Q4 Explain nested if statement with syntax. WAP to find the largest number among three.

Ans: One conditional can also be nested within another

```
if x == y:
print 'x and y are equal'
else:
if x < y:

print 'x is less than y'
else:
print 'x is greater than y'
```

The outer conditional contains two branches. The first branch contains a simple statement. The second branch contains another if statement, which has two branches of its own. Those two branches are both simple statements, although they could have been conditional statements as well.

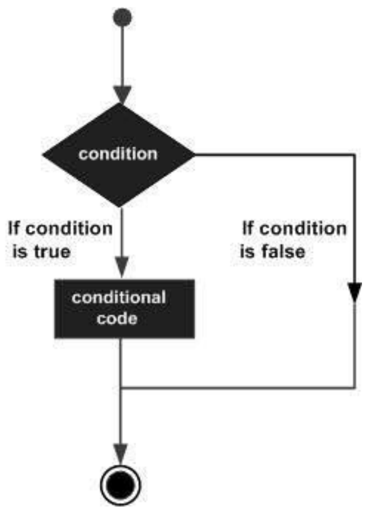
```
# Python program to find the largest number among the three input numbers

num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
num3 = float(input("Enter third number: "))

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

print("The largest number between",num1,",",num2,"and",num3,"is",largest)
```

Q5 How to use control statements in Python.

Ans:

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

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



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

Python programming language provides following types of decision making statements

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

Q6 Ans: List the different operators in python. Explain in detail

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Python Arithmetic Operators

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc. Assume variable a holds 10 and variable b holds 20, then −

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30

- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity):	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

Python Comparison Operators

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

Assume variable a holds 10 and variable b holds 20, then −

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a!=b)
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Python Assignment Operators

Assume variable a holds 10 and variable b holds 20, then −

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Python Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. Assume 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

There are following Bitwise operators supported by Python language

Operator	Description	Example

& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> = 15 (means 0000 1111)

Python Logical Operators

There are following logical operators supported by Python language. Assume variable a holds 10 and variable b holds 20 then

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

Python 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 below

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

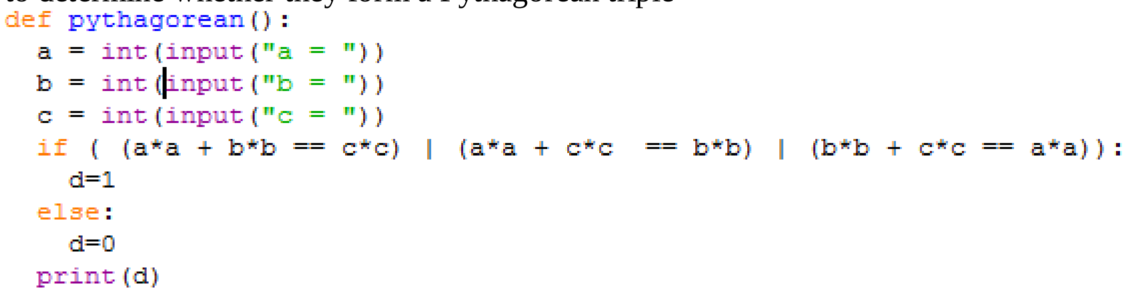
Python Identity Operators

Identity operators compare the memory locations of two objects. There are two Identity operators explained below:

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here is results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here is not results in 1 if id(x) is not equal to id(y).

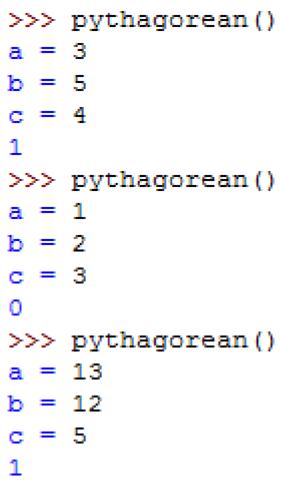
Q7 A triple of numbers (a,b,c) is called a Pythagorean triple if a * a + b * b = c * c. In this question, you will be given three numbers. You have to output 1 if the three numbers form a Pythagorean triple. Otherwise, you have to output 0. Note that the inputs may not be given in order: you may have to try all possible orderings of the three numbers to determine whether they form a Pythagorean triple

Ans:



```
def pythagorean():
  a = int(input("a = "))
  b = int(input("b = "))
  c = int(input("c = "))
  if ( (a*a + b*b == c*c) | (a*a + c*c  == b*b) | (b*b + c*c == a*a)):
    d=1
  else:
    d=0
  print(d)
```

OUTPUT



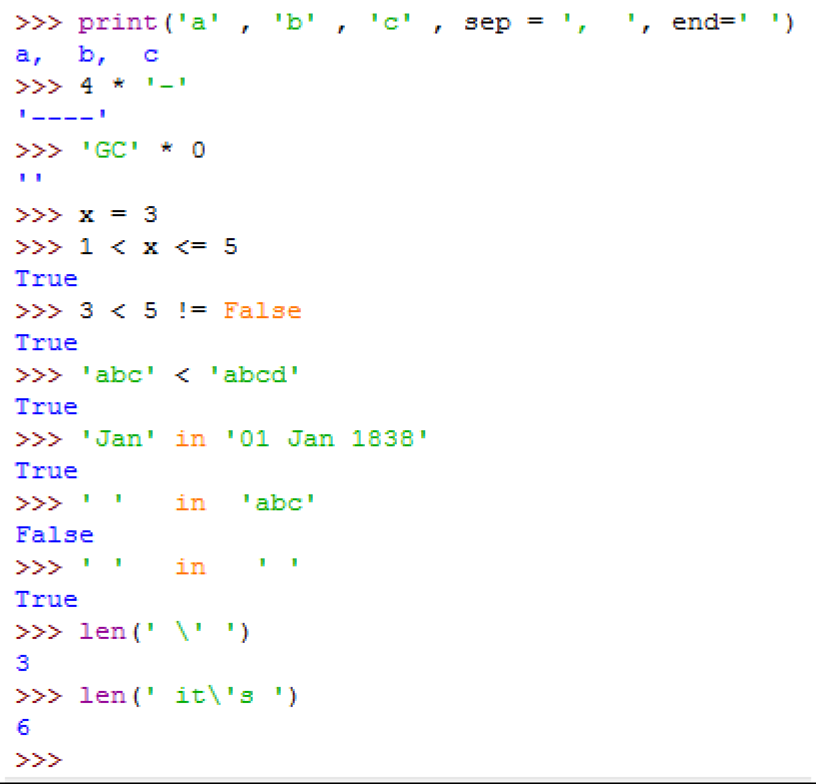
```
>>> pythagorean()
a = 3
b = 5
c = 4
1
>>> pythagorean()
a = 1
b = 2
c = 3
0
>>> pythagorean()
a = 13
b = 12
c = 5
1
```

Q8 Write the output for the following:

i) >>> print('a' , 'b' , 'c' , sep = ', ', end=' ')
ii) >>> 4 * '-'
iii) >>> 'GC' * 0 #This is ‘GC’ * zero
iv) >>> x = 3
    >>> 1 < x <= 5
v) >>> 3 < 5 != False
vi) >>> 'abc' < 'abcd'
vii) >>> 'Jan' in '01 Jan 1838'
viii) >>> ' ' in 'abc'

ix) >>> ' ' in ' '
x) >>> len(' \' ')
>>>len(' it\'s ')

Ans:



```
>>> print('a' , 'b' , 'c' , sep = ',  ', end=' ')
a,  b,  c
>>> 4 * '-'
'----'
>>> 'GC' * 0
''
>>> x = 3
>>> 1 < x <= 5
True
>>> 3 < 5 != False
True
>>> 'abc' < 'abcd'
True
>>> 'Jan' in '01 Jan 1838'
True
>>> ' '   in  'abc'
False
>>> ' '   in   ' '
True
>>> len(' \' ')
3
>>> len(' it\'s ')
6
>>>
```

UNIT 2: Unit name

Q1 How to define and access the function in python?

Ans:

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

Function blocks begin with the keyword **def** followed by the function name and parentheses (( )).

Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.

The first statement of a function can be an optional statement - the documentation string of the function or *docstring*. The code block within every function starts with a colon (:) and is indented.

The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Syntax

def functionname(parameters):
"function_docstring"
function_suite
return [expression]

Example

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

Calling a Function

Defining a function gives it a name, specifies the parameters that are to be included in the

	function and structures the blocks of code. printme(“ram”)	

Q2
Ans:
	Explain any 5 string functions with example.
	
	s.lower(), s.upper() -- returns the lowercase or uppercase version of the string
	s.strip() -- returns a string with whitespace removed from the start and end
	s.isalpha()/s.isdigit()/s.isspace()... -- tests if all the string chars are in the various character classes
	s.replace('old', 'new') -- returns a string where all occurrences of 'old' have been replaced by 'new'
	
	str="hello world"
	
	str2="world"
	
	print(str.split())
	
	print(str.find(str2))
	
	print(str.swapcase())
	
	print(str.lower())
	
	print(str.upper())
	
	print(len(str))
	
	print(str.startswith('h'))
	
	print(str.isdigit())	

Q3 Ans:	What is import? Explain ways of importing in python There are many built in libraries ,modules and packages are provided by python to support easy programming. These all libraries we can use in the code by using import statement. When the interpreter encounters an import statement ,it imports the module ,if the module is present in the search path. There are two ways to use import statement in python:- 1. Import statement Syntax:- import m1[,m2….mN] e.g. import math print (math.sqrt(4)) 2. from….import statement It is used to import specific attributes from a module. Syntax:-from mname import nm1[,nm2…..,nmN] e.g. from math import pi print(pi) 3. from ….import * It is used to import all items from a module. Syntax:-from mname import *	

```
e.g  from math import *
         print(pi)
            print(sqrt(4))
```

Q4
Ans:

WAP to counts the number of times the letter appears in string

```
str=’hello world’

n=0

for i in str:

        if i==’l’:

             n=n+1

       print(n)
```

Q5
Ans:

Explain any 5 Math functions with example

Math functions are as follows:-

math.pow(*x*, *y*)

Return x raised to the power y

math.sqrt(*x*)

Return the square root of *x*.

math.ceil(*x*)
Return the ceiling of *x*, the smallest integer greater than or equal to *x*.

math.fabs(*x*)

Return the absolute value of *x*.

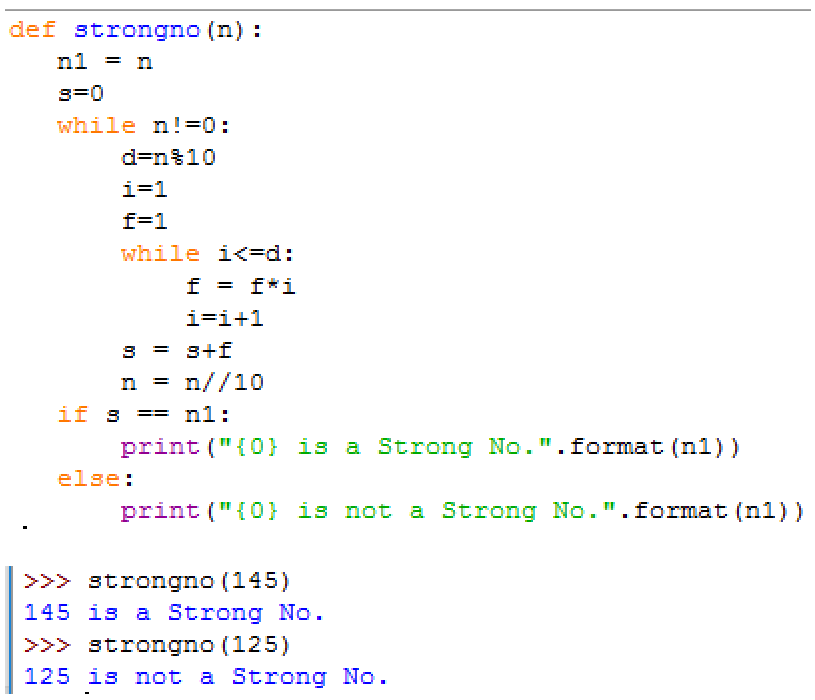
math.factorial(*x*)
Return *x* factorial.

```
import math

print(math.sqrt(4))

print(math.floor(4.1))

print(math.pow(2,3))

print (math.ceil(2.3))

print (math.fabs(-2))

print (math.factorial(5))
```

Q6

Write a Python code to check whether the given number is a strong number. Strong

Ans: Numbers are numbers whose sum of factorial of digits is equal to the original number (Example: **145 = 1! +4! + 5!) .**



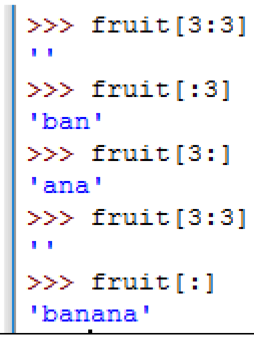
```
def strongno(n):
   n1 = n
   s=0
   while n!=0:
        d=n%10
        i=1
        f=1
        while i<=d:
             f = f*i
             i=i+1
        s = s+f
        n = n//10
   if s == n1:
        print("{0} is a Strong No.".format(n1))
   else:
        print("{0} is not a Strong No.".format(n1))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

```
>>> strongno(145)
145 is a Strong No.
>>> strongno(125)
125 is not a Strong No.
```

Q7 Write the output for the following if the variable **fruit = 'banana'**

Ans:

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



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

Q8 Name the following code and explain the same:

Ans:

i) def recurse():
recurse()

Infinite recursion

If a recursion never reaches a base case, it goes on making recursive calls forever, and the program never terminates. This is known as infinite recursion, and it is generally not a good idea. In most programming environments, a program within finite recursion does not really run forever. Python reports an error message when the maximum recursion depth is reached:

ii) 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)
```

Output:

```
3
2
1
        Blastoff!
```

UNIT 3: Unit name

Q1 Ans:

What are lists? How to define and access the elements of list.

- Like a string,a **list** is a sequence of values.
- In a string, the values are characters; in a list,they can be any type.
- The values in a list are called **elements** or sometimes **items**.
- In a list the elements are enclosed in square brackets ([and]) and separated by commas.
- St=[] :- Empty list
- St1=[1,2,3] :- List of integers
- St2=[11,"abc",1.2] :- List with mixed datatypes
- We can use the index operator[] to access an item in the list. Index starts from 0. So ,a list having 5 elements will have index from 0 to 4.
- Trying to access an element other than this will raise an IndexError. The index must be an integer . Using float or other types will result into TypeError.
- Names=['a','b','c','d','e']
  Names[0]
  o/p:- a
- List can be accessed from both the directions in forward and backward.
- Forward indexing starts with 0,1,2,3
- Backward indexing starts with -1,-2,-3

Q2 Ans:

Explain built-in functions and methods of list.

- Python provides various list functions and list methods to deal with lists.

The various python list functions are:-

a) list():- Converts a tuple into list.
Syntax- list(seq)

	Where seq is a tuple to be converted into list. b) len():- Gives the total length of the list. Syntax-len(lst) Where this is a list for which number of elements to be counted. c) max():- Returns item from the list with max value. Syntax-max(lst) Where lst is maximum valued element to be returned. d) min():-Returns item from the list with min value. Syntax-min(lst) Where lst is minimum valued element to be returned. The various python list functions are:- a) append():- Add an element to the end of the list. Syntax:- list.append(t) Where t is the object to be appended in the list. b) extend():-Add all elements of a list to another list. Syntax:- list.extend(seq) Where seq is the list of elements. c) insert():-Insert an item at the defined index. Syntax:- list.insert(i,t) Where i is the index where the object t need to be inserted And t is the object to be inserted into given list. d) remove():-Removes an item from the list. Syntax:- list.remove(t) Where t is the object to be removed from the list. e) count():-Returns the count of number of items passes as an argument. Syntax:- list.count(t) Where t is the object to be counted in the list.	
Q3 Ans:	Write a short note on tuple operators. The various tuple operators are:- a) Indexing:- The tuple item are indexed. So we can use the index operator [] to access an item in a tuple where the index starts from 0. b) Negative indexing:- The python allows negative indexing for its sequences.The index of -1 refers to the last item ,-2 to the second last item,similarly applied to al other sequence. c) Slicing:- The slicing refers to access the items of selected range. Python	

	provides “:” (colon) operator to achieve this with tuples.	

Q4 Ans:	Explain different file modes and functions used in python.
	• The various file modes supported in python are:-
	a) r :- Open text file for reading.
	b) w :- Open text file for writing.
	c) a:- Open for writing . The file is created if it does not exist.
	d) rb :- Opens a file for reading only in binary format.
	e) wb :- Opens a file for writing in binary format.
	• The various file function supported in python are:-
	a) f.read():-It is used for getting a string containing all characters in the file.
	`f=open(‘t1.txt’,’r’)`
	`print(f.read())`
	b) f.readline():- This function will read line by line from file. It will read a single line from the file and return a string containing characters upto \n.
	`f=open(‘t1.txt’,’r’)`
	`print(f.readline())`
	c) f.write():- This method takes one parameter which is the string to be written.
	`f=open(‘t1.txt’,’w’)`
	`f.write(“hello”)`
	`f.close()`
	d) f.tell():- This method tells you the current cursor position within the file in other words the next read or write will occur at that many bytes from the beginning of the file.
	`f.tell()`
	e) f.seek(offset[,from]):- this method changes the current file position. The offset argument tells the number of bytes to be moved.	

Q5
Ans:

What is directory? Which methods are available to deal with directories in python.

- A directory is a file system cataloging structure which contain references to other computer files, and possibly other directories.
- The python provides various useful built in methods to deal with directories.
- The python os module has several methods that helps to create, remove and change directories by providing several useful functions.

a) os.mkdir("name") :- This method of os module is used to create directories in the current directory by providing name of the directory.

b) os.chdir("dirPathName") :- This method change the working directory by providing the new path of directory as an argument.

c) os.getcwd() :- This method gets the name of current working directory.

d) rmdir("dirName") :- This method is used to remove/delete the directory mentioned in the argument.

e) os.rename(current_dir_name,new_dir_name) :- This method is used to rename existing directory by provoding two arguments as first as the name of the existing file and second as the name that we want to assign.

Q6
Ans:

Explain Exception handling block with example.

- Exception is an abnormal condition in a program code.
- It is an error that happens during execution of a program
- When error occurs, python generate an exception that can be handled , which avoids program to crash.
- One of the way to handle exception is try—except[exception-name]:- with this try block will be executed first and if an exception occurs, the rest of the try block will be skipped and the except clause will be executed.
- Syntax :- code statements

```
except:
 exception handling
```

```
while True:
```

```
          try:
                    x=int(input("Enter a number:"))
                    break
          except ValueError:
                    print("That was no valid number. Try  again..")
```

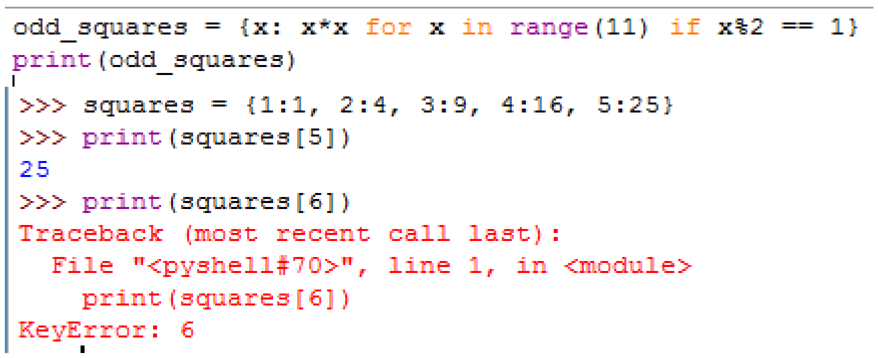
output:-
Enter a number: i
That was no valid number. Try again..
Enter a number: 4

Q7
Ans:

i) What is the significant difference between list and dictionary?
ii) Write a Python code to get the following dictionary as output :
{1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
iii) What is the output of the code given below:

```
>>> squares = {1:1, 2:4, 3:9, 4:16, 5:25}
>>> print(squares[5])
>>> print(squares[6])
```

A dictionary is like a list ,but in a list , the indices have to be integers, in a dictionary they ca
be almost any type.



```
odd_squares = {x: x*x for x in range(11) if x%2 == 1}
print(odd_squares)
```

```
>>> squares = {1:1, 2:4, 3:9, 4:16, 5:25}
>>> print(squares[5])
25
>>> print(squares[6])
Traceback (most recent call last):
  File "<pyshell#70>", line 1, in <module>
    print(squares[6])
KeyError: 6
```

Q8
Ans:

Write the output for the following:

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


        [1, 2, 3, 4, 5, 6]



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

        [1, 2, 3, 1, 2, 3, 1, 2, 3]



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


['a', 'x', 'y', 'd', 'e', 'f']
```

UNIT 4: Unit name

Q1 Explain regular expression in python

Ans: A *regular expression* is a special sequence of characters that helps us match or find other strings or sets of strings. The module **re** provides full support for Perl-like regular expressions in Python. The re module raises the exception re.error if an error occurs while compiling or using a regular expression.

1) The match Function

This function attempts to match RE *pattern* to *string* with optional *flags*.

re.match(pattern, string, flags=0)

pattern-This is the regular expression to be matched.

String-This is the string, which would be searched to match the pattern at the beginning of string.

Flags-we can specify different flags using bitwise OR (|). These are modifiers, which are listed in the table below.

group(num=0)- This method returns entire match (or specific subgroup num)

groups()-This method returns all matching subgroups in a tuple (empty if there weren't any)

The *re.match* function returns a **match** object on success, **None** on failure.

2) The *search* Function

This function searches for first occurrence of RE *pattern* within *string* with optional *flags*.

re.search(pattern, string, flags=0)

pattern-This is the regular expression to be matched.

String-This is the string, which would be searched to match the pattern anywhere in the

string.
Flags-We can specify different flags using bitwise OR (|). These are modifiers, which are listed in the table below.

group(num=0)- This method returns entire match (or specific subgroup num)

groups()-This method returns all matching subgroups in a tuple (empty if there weren't any)

Q2 Ans: Define Inheritance. Explain Single inheritance with e.g

It refers to defining a new class with little or no modification to an existing class. The new class is called **derived (or child) class** and the one from which it inherits is called the **base (or parent) class**.

```
class BaseClass:
  Body of base class
class DerivedClass(BaseClass):
  Body of derived class
```

EXAMPLE

```
class Person:

    def __init__(self, first, last):
        self.firstname = first
        self.lastname = last

    def Name(self):
        return self.firstname + " " + self.lastname

class Employee(Person):

    def __init__(self, first, last, staffnum):
        Person.__init__(self,first, last)
        self.staffnumber = staffnum

    def GetEmployee(self):
        return self.Name() + ", " +  self.staffnumber

x = Person("Sam", "Taylor")
y = Employee("Homer", "Simpson", "1007")

print(x.Name())
print(y.GetEmployee())
```

OUTPUT

```
Sam Taylor
Homer Simpson, 1007
```

Q3 How to initialize and delete class object.

Ans:

The method used for initialization is called __init__, which is a special method. When we call the class object, a new instance of the class is created, and the __init__ method on this new object is immediately executed with all the parameters that we passed to the class object. The purpose of this method is thus to set up a new object using data that we have provided. self is the first parameter, and we use this variable inside the method bodies – but we don't appear to pass this parameter in. This is because whenever we call a method on an object, *the object itself* is automatically passed in as the first parameter. This gives us a way to access the object's properties from inside the object's methods.

```
# inside class Time:
def __init__(self, hour=0, minute=0, second=0):
self.hour = hour
self.minute = minute
self.second = second
```

For deleting objects- Any attribute of an object can be deleted anytime, using the del statement.

```
class Vehicle:
    def __init__(self):
        print('Vehicle created.')

    def __del__(self):
        print('Destructor called, vehicle deleted.')

car = Vehicle()
del car
```

OUTPUT

```
Vehicle created.
Destructor called, vehicle deleted.
```

Q4 Ans: Explain the methods of achieving synchronization of multithreaded programs in Python.

The threading module provided with Python includes a simple-to-implement locking mechanism that allows you to synchronize threads. A new lock is created by calling the **Lock()** method, which returns the new lock.

The **acquire(blocking)** method of the new lock object is used to force threads to run synchronously. The optional *blocking* parameter enables you to control whether the thread waits to acquire the lock.

If *blocking* is set to 0, the thread returns immediately with a 0 value if the lock cannot be acquired and with a 1 if the lock was acquired. If blocking is set to 1, the thread blocks and wait for the lock to be released.

The **release()** method of the new lock object is used to release the lock when it is no longer required.

Example

```
import threading import time
class myThread (threading.Thread):
def init (self, threadID, name, counter): threading.Thread. init (self) self.threadID =
threadID
self.name = name self.counter = counter
def run(self):
print ("Starting " + self.name)
```

```
# Get lock to synchronize threads threadLock.acquire() print_time(self.name, self.counter, 3)
# Free lock to release next thread threadLock.release()
def print_time(threadName, delay, counter): while counter:
time.sleep(delay)
print ("%s: %s" % (threadName, time.ctime(time.time()))) counter -= 1
threadLock = threading.Lock() threads = []
# Create new threads
thread1 = myThread(1, "Thread-1", 1) thread2 = myThread(2, "Thread-2", 2)
# Start new Threads thread1.start() thread2.start()
# Add threads to thread list threads.append(thread1) threads.append(thread2)
# Wait for all threads to complete for t in threads:
t.join()
print ("Exiting Main Thread")
```

OUTPUT

```
Starting Thread-1Starting Thread-2
Thread-1: Sun Jun 18 13:50:07 2017
Thread-1: Sun Jun 18 13:50:08 2017
Thread-1: Sun Jun 18 13:50:09 2017
Thread-2: Sun Jun 18 13:50:11 2017
Thread-2: Sun Jun 18 13:50:13 2017
Thread-2: Sun Jun 18 13:50:15 2017 Exiting Main Thread
```

Q5 Explain different threading functions.

Ans: 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

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

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

The *threading* module exposes all the methods of the *thread* module and provides some additional methods:

- **threading.activeCount():** Returns the number of thread objects that are active.
- **threading.currentThread():** Returns the number of thread objects in the caller's thread control.

- **threading.enumerate():** Returns a list of all thread objects that are currently active.

The methods provided by the *Thread* class are as follows:

- **run():** The run() method is the entry point for a thread.
- **start():** The start() method starts a thread by calling the run method.
- **join([time]):** The join() waits for threads to terminate.
- **isAlive():** The isAlive() method checks whether a thread is still executing.
- **getName():** The getName() method returns the name of a thread.
- **setName():** The setName() method sets the name of a thread.

Q6 Ans: Create a module armprime that has two functions, one to check whether the given number is an Armstrong number and the other to check whether the given number is a prime number. Import the above created module in a .py file and show the use of the two functions.

```
def armstrong(n):
    n1 = n
    s=0
    while n!= 0:
      d = n%10
      s = s+d**3
      n = n//10
    if n1 == s:
       return True
    else:
       return False

def prime(n):
    i=2
    while i <= n-1:
         if n%i == 0:
             return False
         i=i+1
    if n == i:
        return True

import armprime as ap
a = ap.armstrong(153)
print(a)
b=ap.prime(7)
print(b)
```

Output:-

```
True
```

True

Q7 Ans:

Explain the role of the Regular Expressions in the following snippets:

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')

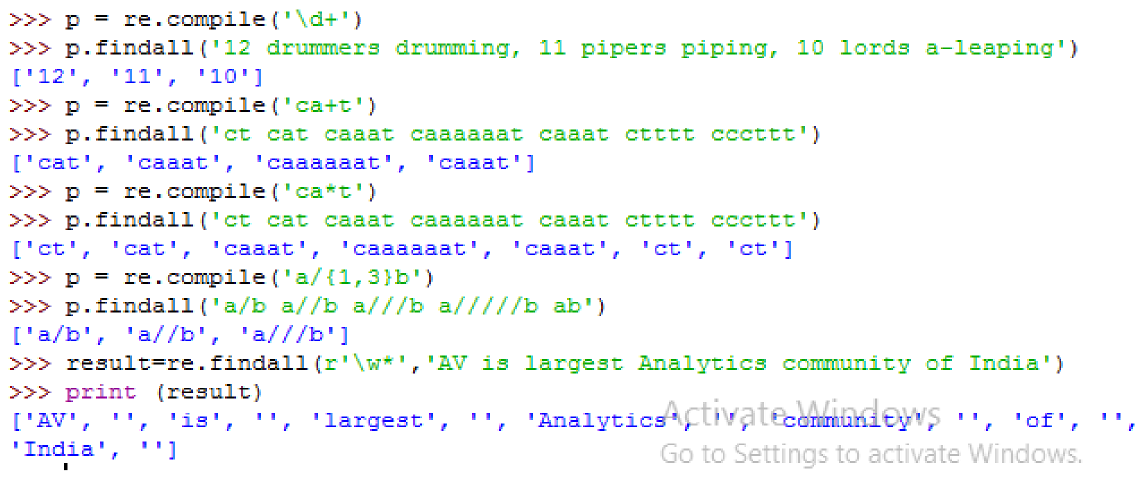
>>> p.findall('ct cat caaat caaaaaat caaat ctttt cccttt')

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)



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

Q8 Ans:

How is method overriding implemented in Python?

class Parent(object):
def __init__(self):
self.value = 5
def get_value(self):
return self.value
class Child(Parent):
def get_value(self):
return self.value + 1
c=Child()
print(c.get_value())

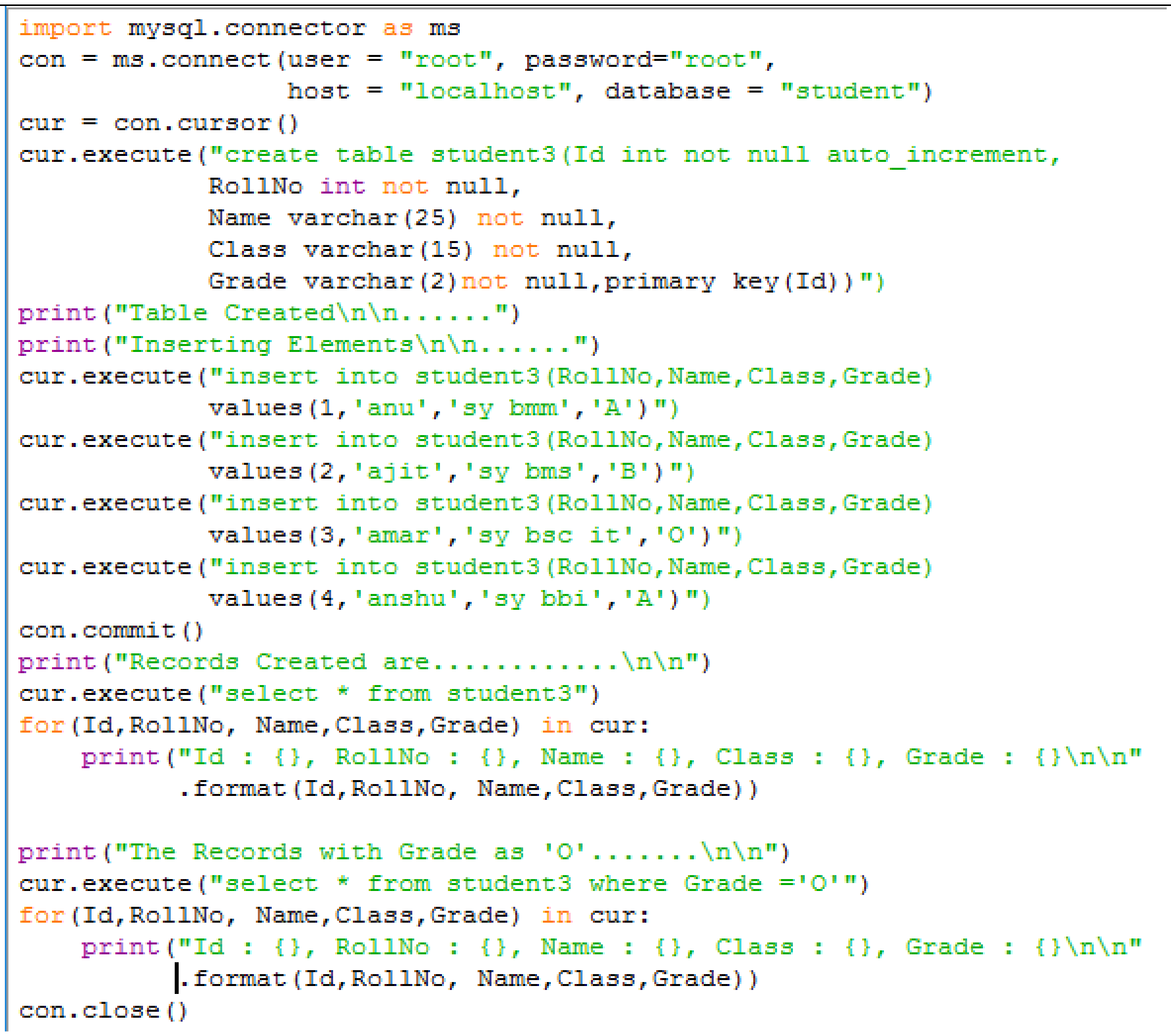
Output:

6

UNIT 5: Unit name

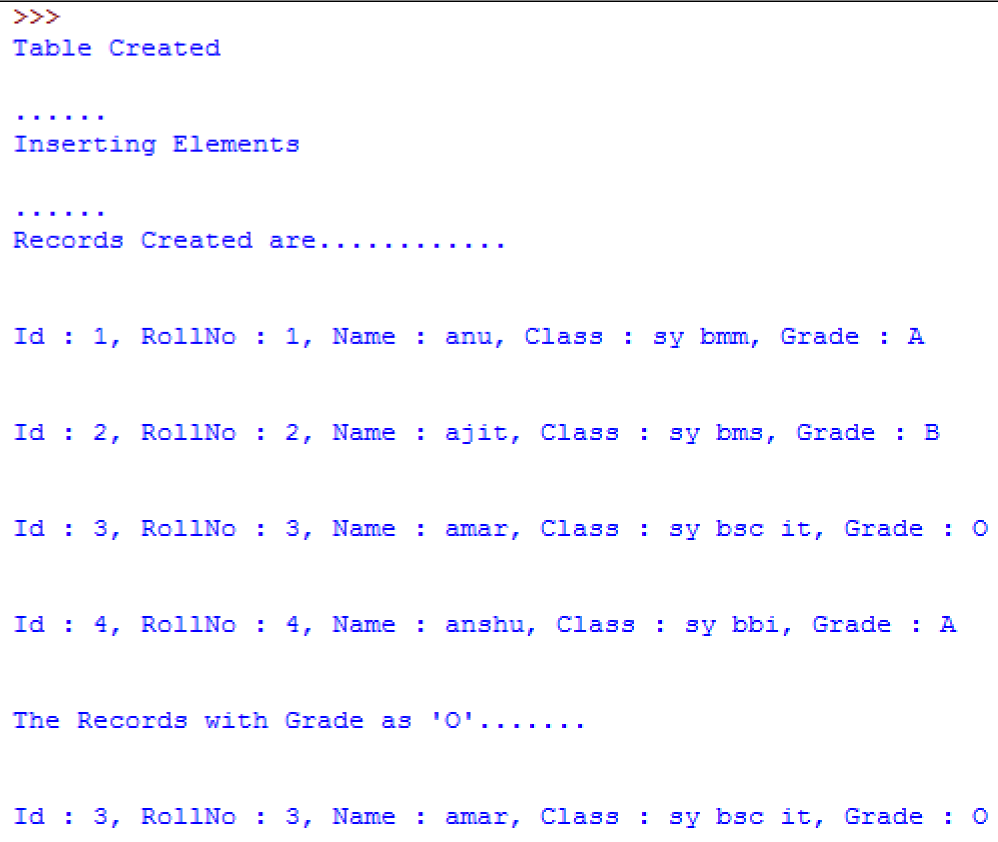
Q1 Ans:

Write a Python code to do the following: i) Create a table STUDENT with columns Id (integer, not null, primary key), RollNo (integer, not null), Name (not null, varchar), Class (not null, varchar) and Grade (not null, varchar) in MySQL and insert 4 rows of data. ii)Retrieve the rows where Grade = ‘O’



```python
import mysql.connector as ms
con = ms.connect(user = "root", password="root",
                 host = "localhost", database = "student")
cur = con.cursor()
cur.execute("create table student3(Id int not null auto_increment,
            RollNo int not null,
            Name varchar(25) not null,
            Class varchar(15) not null,
            Grade varchar(2)not null,primary key(Id))")
print("Table Created\n\n......")
print("Inserting Elements\n\n......")
cur.execute("insert into student3(RollNo,Name,Class,Grade)
            values(1,'anu','sy bmm','A')")
cur.execute("insert into student3(RollNo,Name,Class,Grade)
            values(2,'ajit','sy bms','B')")
cur.execute("insert into student3(RollNo,Name,Class,Grade)
            values(3,'amar','sy bsc it','O')")
cur.execute("insert into student3(RollNo,Name,Class,Grade)
            values(4,'anshu','sy bbi','A')")
con.commit()
print("Records Created are............\n\n")
cur.execute("select * from student3")
for(Id,RollNo, Name,Class,Grade) in cur:
    print("Id : {}, RollNo : {}, Name : {}, Class : {}, Grade : {}\n\n"
          .format(Id,RollNo, Name,Class,Grade))

print("The Records with Grade as 'O'.......\n\n")
cur.execute("select * from student3 where Grade ='O'")
for(Id,RollNo, Name,Class,Grade) in cur:
    print("Id : {}, RollNo : {}, Name : {}, Class : {}, Grade : {}\n\n"
          .format(Id,RollNo, Name,Class,Grade))
con.close()
```

OUTPUT:



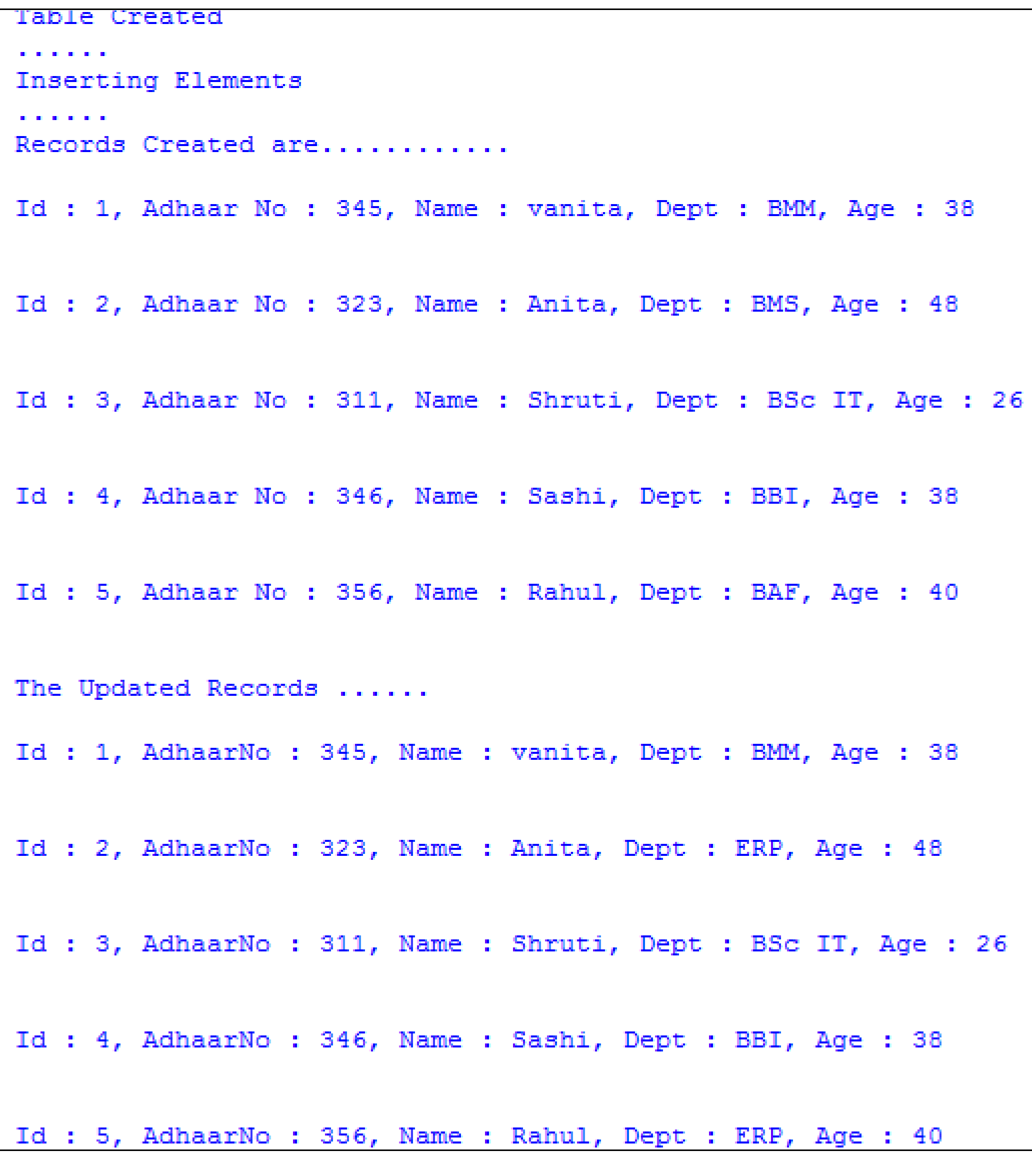
```
>>>
Table Created

......
Inserting Elements

......
Records Created are............


Id : 1, RollNo : 1, Name : anu, Class : sy bmm, Grade : A


Id : 2, RollNo : 2, Name : ajit, Class : sy bms, Grade : B


Id : 3, RollNo : 3, Name : amar, Class : sy bsc it, Grade : O


Id : 4, RollNo : 4, Name : anshu, Class : sy bbi, Grade : A


The Records with Grade as 'O'.......


Id : 3, RollNo : 3, Name : amar, Class : sy bsc it, Grade : O
```

Q2
Ans:

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.

```python
import mysql.connector as ms
con = ms.connect(user = "root", password="root",
                 host = "localhost", database = "student")
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) not null,
            Empage int not null,primary key(Id))")
print("Table Created\n\n......")
print("Inserting Elements\n\n......")
cur.execute("insert into emp2(AdhaarNo,Empname,Empdept,Empage)
            values(345,'vanita','BMM',38)")
cur.execute("insert into emp2(AdhaarNo,Empname,Empdept,Empage)
            values(323,'Anita','BMS',48)")
cur.execute("insert into emp2(AdhaarNo,Empname,Empdept,Empage)
            values(311,'Shruti','BSc IT',26)")
cur.execute("insert into emp2(AdhaarNo,Empname,Empdept,Empage)
            values(346,'Sashi','BBI',38)")
cur.execute("insert into emp2(AdhaarNo,Empname,Empdept,Empage)
            values(356,'Rahul','BAF',40)")
con.commit()
print("Records Created are............\n\n")
cur.execute("select * from emp2")
for(Id,AdhaarNo,Empname,Empdept,Empage) in cur:
    print("Id : {}, Adhaar No : {}, Name : {}, Dept : {}, Age : {}\n\n"
          .format(Id,AdhaarNo,Empname,Empdept,Empage))

print("The Updated Records ......\n\n")
cur.execute("update emp2 set Empdept = 'ERP'  where Empage >=40")
cur.execute("select * from emp2")
for(Id,AdhaarNo,Empname,Empdept,Empage) in cur:
    print("Id : {}, AdhaarNo : {}, Name : {}, Dept : {}, Age : {}\n\n"
          .format(Id,AdhaarNo,Empname,Empdept,Empage))
con.close()
```

OUTPUT:-



```
Table Created
......
Inserting Elements
......
Records Created are............

Id : 1, Adhaar No : 345, Name : vanita, Dept : BMM, Age : 38


Id : 2, Adhaar No : 323, Name : Anita, Dept : BMS, Age : 48


Id : 3, Adhaar No : 311, Name : Shruti, Dept : BSc IT, Age : 26


Id : 4, Adhaar No : 346, Name : Sashi, Dept : BBI, Age : 38


Id : 5, Adhaar No : 356, Name : Rahul, Dept : BAF, Age : 40


The Updated Records ......

Id : 1, AdhaarNo : 345, Name : vanita, Dept : BMM, Age : 38


Id : 2, AdhaarNo : 323, Name : Anita, Dept : ERP, Age : 48


Id : 3, AdhaarNo : 311, Name : Shruti, Dept : BSc IT, Age : 26


Id : 4, AdhaarNo : 346, Name : Sashi, Dept : BBI, Age : 38


Id : 5, AdhaarNo : 356, Name : Rahul, Dept : ERP, Age : 40
```

Q3 Explain following widgets used in python

Ans: a)text widget b)button c)label

Text Widget :-

This widget allows to display and edit text with various styles and attributes.

It also supports embedded images and windows.

It stores and displays lines of text.

The text body can have characters, embedded windows or images.

By default the text widget's contents can be edited using standard keyboard and mouse bindings. If editing is not needed then DISABLE option is available.

Example:-
From tkinter import *
Root=TK()
T=Text(root,height=3,width=50)
T.pack()
T.insert(END,"hello\neveryone\n")
mainloop()

Button:-

- This widget is used to implement various kinds of buttons.
- It can contain text or images.
- When the button is clicked, it automatically calls that function or method associated with it.

Example:-

```
from tkinter import *
root=Tk()
def callback():
      print(“click”)
b=Button(root,text=”ok”,command=callback,bg=”Red”fg=”Green”)
b.pack()
mainloop()
```

LABEL:-

- This widget is used to display text or an image.
- it is only to display the content where user can just view but cannot interact with the content.
- It is basically used to give information to the user.

Example:-

```
From tkinter import *
Root=Tk()
L=Label(root,text=”Hello world”)
l.pack()
root.mainloop()
```

Q4 Ans: Write a short note on the creation of a menu control in tkinter

- We can display all the commands and functions of the application to the user with help of Menu.
- Menus can be presented with a combination of text and symbols to represent the options.

User can select menu to perform some action related to it.

Example:-

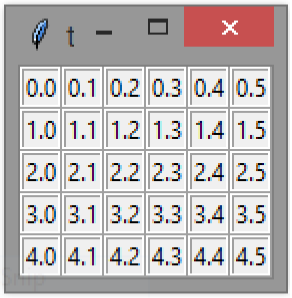
```
From tkinter import Tk,Menu
root=Tk()
menu_bar=Menu(root)
root.config(menu=menu_bar)
file_menu=Menu(menu_bar,tearoff=0)
menu_bar.add_cascade(label=”File”,menu=file_menu)
file_menu.add_command(label=”Quit”,command=root.destroy)
file_menu.add_separator()
file_menu.add_command(label=”Exit”,command=root.destroy)
file_menu.add_separator()
file_menu.add_command(label=”End”,command=root.destroy)
file_menu.add_separator()
root.mainloop()
```

Q5
Ans: Describe the method of enhancing the look and feel of GUI using different appearances of widgets.

The look and feel customization refers to modifying (customizing) the existing standard widgets as per the users requirement for better enhanced and user friendly display.
The python tkinter module provides several widgets with several options and attributes that can be customized as per users needs.

For Example the program shows the use of labels with raised border for better display.

```
from tkinter import *
for i in range(5):
        for j in range(6):
                l=Label(text='%d.%d'%(i,j),relief=RIDGE)
                l.grid(row=i,column=j,sticky=NSEW)
mainloop()
```



```
0.0 0.1 0.2 0.3 0.4 0.5
1.0 1.1 1.2 1.3 1.4 1.5
2.0 2.1 2.2 2.3 2.4 2.5
3.0 3.1 3.2 3.3 3.4 3.5
4.0 4.1 4.2 4.3 4.4 4.5
```

Q6
Ans: Write a short note on Listbox widget of tkinter with an example.

The Listbox widget is used to display a list of items from which a user can select a number of items.

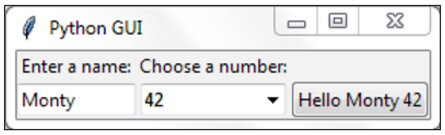
Syntax:-w=listbox(master,option,..)

```
from Tkinter import *

import tkMessageBox

import Tkinter

top = Tk()
```

```
Lb1 = Listbox(top)

Lb1.insert(1, "Python")

Lb1.insert(2, "Perl")

Lb1.insert(3, "C")

Lb1.insert(4, "PHP")

Lb1.insert(5, "JSP")

Lb1.insert(6, "Ruby")

Lb1.pack()

top.mainloop()
```

Q7
Ans:

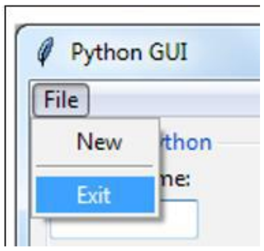
Write a Python code to create the following GUI:

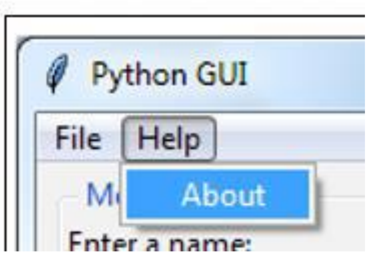


```
import tkinter as tk
from tkinter import ttk

win = tk.Tk()
win.title("Python GUI")

# Modified Button Click Function
def clickMe():
    action.configure(text='Hello ' + name.get()+ ' ' +
    numberChosen.get())
# Position Button in second row, second column(zero-based)
action = ttk.Button(win, text="Click Me!", command=clickMe)
action.grid(column=2, row=1)
# Label
ttk.Label(win, text="Enter a name:").grid(column=0, row=0)
# Adding a Textbox Entry widget
name = tk.StringVar()
nameEntered = ttk.Entry(win, width=12, textvariable=name)
nameEntered.grid(column=0, row=1)

ttk.Label(win, text="Choose a number:").grid(column=1, row=0)
number = tk.StringVar()
numberChosen = ttk.Combobox(win, width=12, textvariable=number)
numberChosen['values'] = (1, 2, 4, 42, 100)
numberChosen.grid(column=1, row=1)
numberChosen.current(0)

win.mainloop()
```

Q8
Ans:

Write a Python code to create the following menu bar :





```
import tkinter as tk
from tkinter import ttk
from tkinter import Menu

win = tk.Tk()
win.title("Python GUI")
menuBar = Menu(win)
win.config(menu=menuBar)
fileMenu = Menu(menuBar)
fileMenu.add_command(label="New")
fileMenu.add_separator()
fileMenu.add_command(label="Exit")
menuBar.add_cascade(label="File", menu=fileMenu)
fileMenu = Menu(menuBar, tearoff=0)

helpMenu = Menu(menuBar, tearoff=0)
helpMenu.add_command(label="About")
menuBar.add_cascade(label="Help", menu=helpMenu)

win.mainloop()
```