1. (a) Write a python program to find the area of square, rectangle and circle. Print the results. Take input from user 6

   (b) List and explain the syntax of all flow control statements with example. 8
   Flow Control Statements: if Statements

- An if statement's clause (the block following the if statement) will execute if the statement's condition is True. The clause is skipped if the condition is False.
- An if statement could be read as, "If this condition is true, execute the code in the clause."
- In Python, an if statement consists of the following:
  - The if keyword
  - A condition (that is, an expression that evaluates to True or False)
  - A colon
  - Starting on the next line, an indented block of code (called the if clause)
- Ex:-
-     if name == 'Alice':
-             print('Hi Alice')
- **Flow Control Statements: else Statements**
- An if clause can optionally be followed by an else statement. The else clause is executed only when the if statements condition is False.
- An else statement could be read as, "If this condition is true, execute this code. Or else, execute that code."
- An else statement doesn't have a condition, and in code, an else statement
- always consists of the following:
  - The else keyword
  - A colon
  - Starting on the next line, an indented block of code (called the else clause)
- Ex:-
-     if name == 'Alice':
-             print('Hi Alice')
-     else:
-              print('Hello Stranger')
- **Flow Control Statements: elif Statements**
- The elif statement is an "else if" statement that always follows an if or another elif statement.
- It provides another condition that is checked only if any of the previous conditions were False.
- In code, an elif statement always consists of the following:
  - The elif keyword
  - A condition (that is, an expression that evaluates to True or False)
  - A colon
  - Starting on the next line, an indented block of code (called the elif clause)
- if name == 'Alice':
-         print('Hi, Alice.')
- elif age < 12:
-         print('You are not Alice, kiddo.')

```
if name == 'Alice':
        print('Hi, Alice.')
elif age < 12:
        print('You are not Alice, kiddo.')
elif age > 100:
        print('You are not Alice, grannie.')
```

Flow Control Statements: while Loop Statements

- The code in a while clause will be executed as long as the while statement's condition is True.
- In code, a while statement always consists of the following:
  - ❖ The while keyword
  - ❖ A condition (that is, an expression that evaluates to True or False)
  - ❖ A colon
  - ❖ Starting on the next line, an indented block of code (called the while clause)

```
• spam = 0
• while spam < 5:
•         print('Hello, world.')
•         spam = spam + 1
```

(c) Illustrate the use of break and continue with a code snippet. 6

- **Flow Control Statements: break Statements**
- If the execution of the program reaches a break statement it immediately exits the while loop's clause.

```
• while True :
•         print('Enter name of the subject')
•      name = input()
•      if name == 'python':
•            break
• print('Thanks')
```

2. (b) Explain global statement with example. 8

   1. To modify a global variable from within a function, use the global statement.
   2. If you have a line such as global eggs at the top of a function, it tells Python, "In this function, eggs refers to the global variable, so don't create a local variable with this name.

```
def spam():
  global eggs              #global keyword helps in modifying global variable
  eggs = 'spam'       #value inside function
eggs = 'global'              #eggs is global variable

spam()

print(eggs)                          #outputs updated global value i.e. spam
```

- There are four rules to tell whether a variable is in a local scope or global scope:
    1. If a variable is being used in the global scope (that is, outside of all functions), then it is always a global variable.
    2. If there is a global statement for that variable in a function, it is a global variable.
    3. Otherwise, if the variable is used in an assignment statement in the function, it is a local variable.
    4. But if the variable is not used in an assignment statement, it is a global variable.
- **def spam():**
- **global eggs**
- **eggs = 'spam'   # this is the global**
- **def bacon():**
- **eggs = 'bacon'   # this is a local**
- **def ham():**
- **print(eggs)   # this is the global**
- **eggs = 42   # this is the global**
- **spam()**
- **print(eggs)   #outputs spam**
- If you try to use a local variable in a function before you assign a value to it, as in the following program, Python will give you an error.
- **def spam():**
- **print(eggs) # ERROR!**
- **eggs = 'spam local'**
- **eggs = 'global'**
- **spam()   #Raises Error**

3.(a) What is list?Explain the concept of list slicing with example. 6

- **A List is a data structure that contains multiple values in ordered sequence.**
- **Example : [‘mango’, ‘apple’, ‘pineapple’, ‘banana’]**
- **A slice can get several values from a list in the form of a new list. A slice is typed between square brackets like an index but it has 2 integers separated by a colon.**

- **In a slice the first integer is the index where the slice starts. The second integer is the index where the slice ends. A slice goes upto but will not include the value at the second index.**
- **spam = [‘cat’, ‘bat’, ‘rat’, ‘pat’]**

spam[0:4] # [‘cat’, ‘bat’, ‘rat’, ‘pat’]

spam[1:3]   #   [‘bat’, ‘rat’]

spam[0:-1]   #   [‘cat’, ‘bat’, ‘rat’]

- **We can leave out one or both of the indexes on either side of the colon in the slice. Leaving out the first index is the same as using 0. Leaving out the second index is same as using the length of the list.**
- **spam[:2]   #   [‘cat’, ‘bat’]**

spam[1:]   #   [‘bat’, ‘rat’, ‘pat’]

spam[:] #   [‘cat’, ‘bat’ , ‘rat’, ‘pat’]

- **Lists length: The len() function will return the number of values that are in a list passed to it.**

Ex:- spam = [‘cat’, ‘dog’, ‘moose’]

len(spam)   #   3

3. (b) Explain references with example. 7

Consider the following example.

spam= 42

cheese = spam

spam = 100

The above lines of code

1. **Assigns 42 to spam variable.**
2. **The value in spam is copied and assigned to cheese variable.**
3. **Assigns 100 to spam variable. The cheese value remains unchanged because spam and cheese are different variables.**

When we assign a list to a variable we are actually assigning a list reference to a variable.

A reference is a value that points to some bit of data.

A list reference is a value that points to a list.

When we assign a list to a variable we are actually assigning a list reference to a variable.

A reference is a value that points to some bit of data.

A list reference is a value that points to a list.

Consider the following example.

spam= [0,1,2,3,4,5]

cheese = spam

cheese[1] = ‘Hello!’ # [0, ‘Hello!’,2,3,4,5]

cheese # [0, ‘Hello!’,2,3,4,5]

Spam   # [0, ‘Hello!’,2,3,4,5]

Python uses references whenever variable must store values of mutable data types such as lists or dictionaries.

For values of immutable data types such as strings, integers or tuples python variable will store the value itself.

When a function accepts list as an argument a copy of the list reference is passed.

Ex:-

def eggs(someparameter):

someparameter.append(‘Hello’)

spam = [1,2,3]

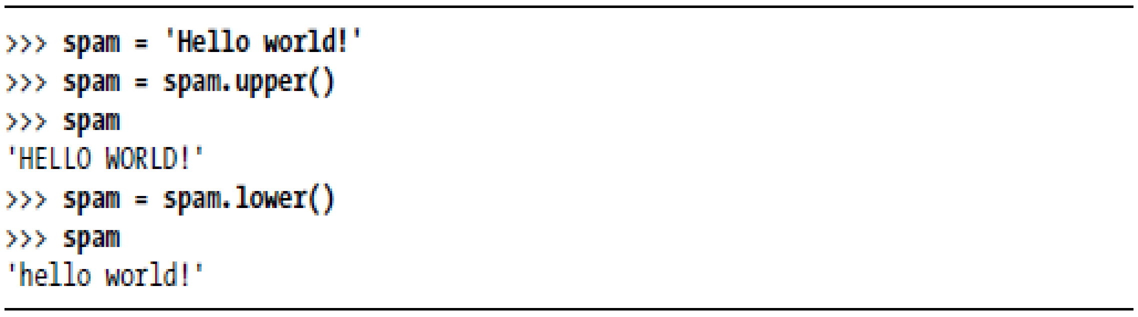
eggs(spam)

print(spam)   # [1,2,3, ‘Hello’]

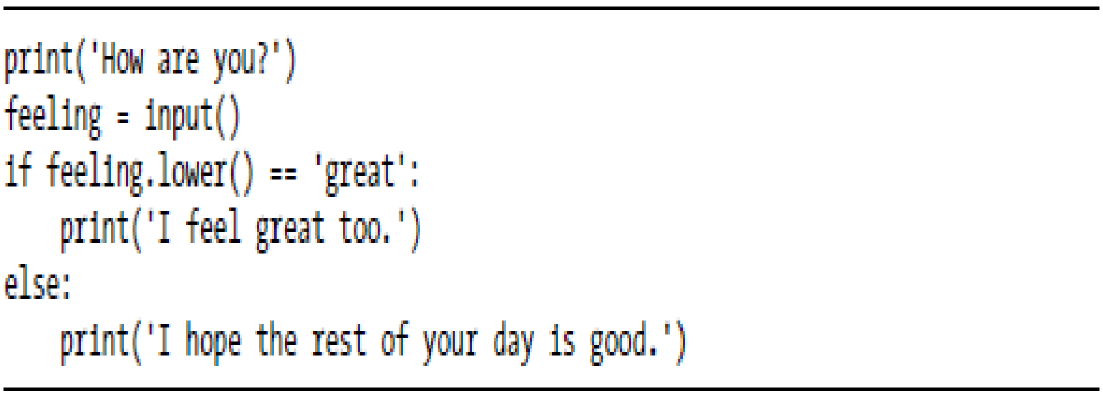
Since list reference is passed to function any changes to list in function will be reflected outside the function also.

4. (b) List any six methods associated with string and explain each of them with example. 8

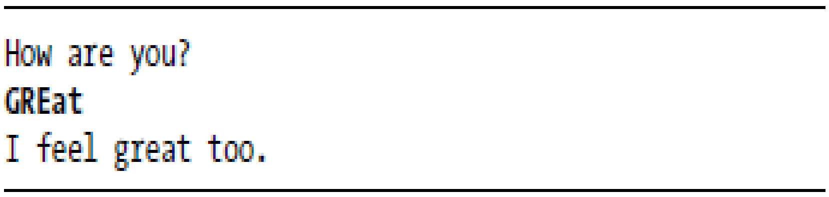
Useful String methods upper(), lower(), isupper(), and islower()



```
>>> spam = 'Hello world!'
>>> spam = spam.upper()
>>> spam
'HELLO WORLD!'
>>> spam = spam.lower()
>>> spam
'hello world!'
```



```
print('How are you?')
feeling = input()
if feeling.lower() == 'great':
    print('I feel great too.')
else:
    print('I hope the rest of your day is good.')
```

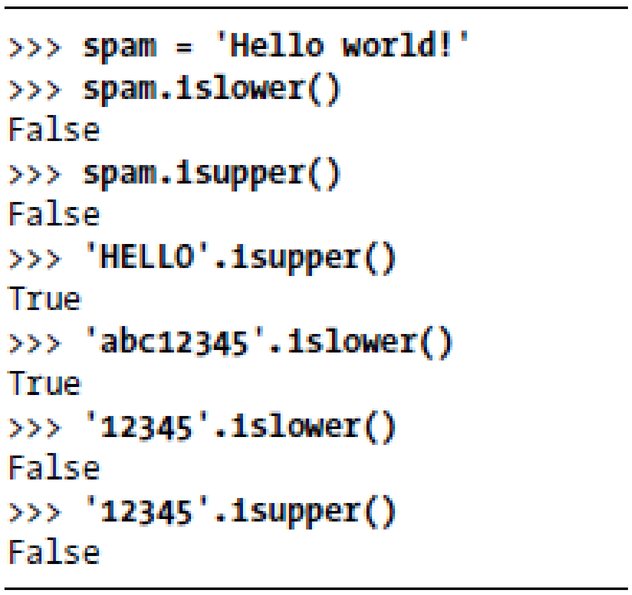


```
How are you?
GREat
I feel great too.
```

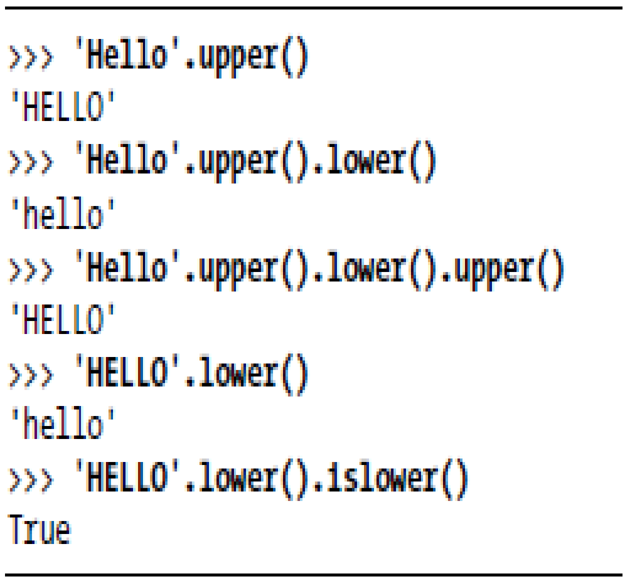
Useful String methods upper(), lower(), isupper(), and islower()

The isupper() and islower() methods will return a Boolean True value

if the string has at least one letter and all the letters are uppercase or lowercase, respectively.



```
>>> spam = 'Hello world!'
>>> spam.islower()
False
>>> spam.isupper()
False
>>> 'HELLO'.isupper()
True
>>> 'abc12345'.islower()
True
>>> '12345'.islower()
False
>>> '12345'.isupper()
False
```



```
>>> 'Hello'.upper()
'HELLO'
>>> 'Hello'.upper().lower()
'hello'
>>> 'Hello'.upper().lower().upper()
'HELLO'
>>> 'HELLO'.lower()
'hello'
>>> 'HELLO'.lower().islower()
True
```

isX string methods

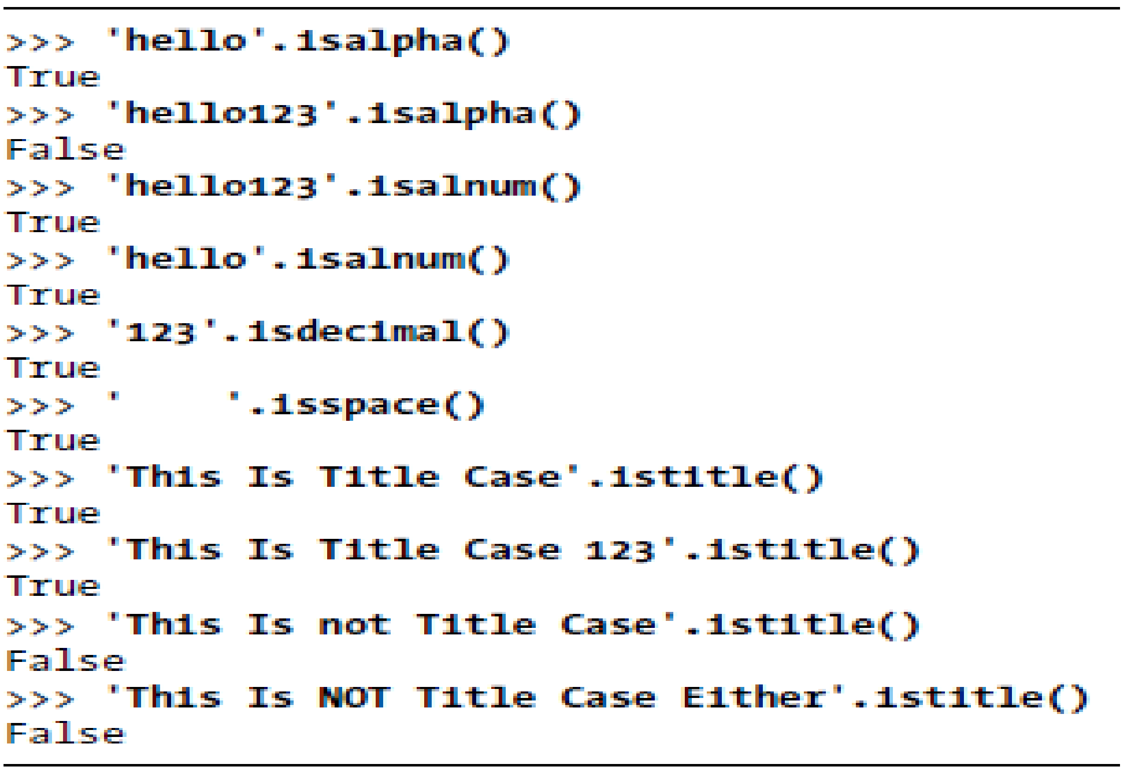
isalpha() returns True if the string consists only of letters and is not blank.

isalnum() returns True if the string consists only of letters and numbers and is not blank.

isdecimal() returns True if the string consists only of numeric characters and is not blank.

isspace() returns True if the string consists only of spaces, tabs, and newlines and is not blank.

istitle() returns True if the string consists only of words that begin with an uppercase letter followed by only lowercase letters.



```
>>> 'hello'.isalpha()
True
>>> 'hello123'.isalpha()
False
>>> 'hello123'.isalnum()
True
>>> 'hello'.isalnum()
True
>>> '123'.isdecimal()
True
>>> '     '.isspace()
True
>>> 'This Is Title Case'.istitle()
True
>>> 'This Is Title Case 123'.istitle()
True
>>> 'This Is not Title Case'.istitle()
False
>>> 'This Is NOT Title Case Either'.istitle()
False
```

5. (a) List and explain Shorthand code for common character classes .Illustrate how do you define your own character class? 7

Shorthand character class	**Represents**
\d	**Any numeric digit from 0 to 9**
\D	**Any character that is not a numeric dig**
\w	**Any letter, numeric digit or the unders**
\W	**Any character that is not a letter, nume underscore character.**
\s	**Any space or tab or newline character**
\S	**Any character that is not a space or tab**

import re

regex = re.compile(r'\d+\s\w+')

lists = regex.findall('12 drummers, 11 pipers, 10 lords, 9 ladies, 8 maids, 7

swans, 6 geese, 5 rings, 4 birds, 3 hens, 2 doves, 1 partridge')

print(lists)

Making own Character classes

We can define our Own character class using square brackets.

For Ex:- the character class [aeiouAEIOU] will match any vowel both upper and lower cases.

```
import re

regex = re.compile(r'[aeiouAEIOU]')

lists = regex.findall('RoboCop eats baby food. BABY FOOD.')
```

Inside square brackets the normal regular expression symbols are not interpreted as such. We do not need to escape ., *, ? or ()

6. (b) Explain the usage of Caret and dollar sign characters in regular expression. 6

If ^ sign is used at the start of a regex to indicate that a match must occur at the beginning of the searched text.

If $ sign is used at the end of a regex to indicate that the string must end with regex pattern.

If ^ and $ signs are used at the start and end of a regex the entire string must match the regex.

```
import re

regex = re.compile(r'^Hello')

match = regex.search('Hello world!')

print(match.group())

match1 = regex.search('He said Hello')

if match1 is None:

    print(‘No match is found')
```

```
import re

regex = re.compile(r‘\d$')

match = regex.search('Your number is 42')

print(match.group())

match1 = regex.search('Your number is forty two.')

if match1 is None:

    print('No match is found')
```

```
import re
regex = re.compile(r'^\d+$')
match = regex.search('1234567890')
print(match.group())
match1 = regex.search('12345xyz67890')
if match1 is None:
    print('No match is found')
match2 = regex.search('12 34567890')
if match2 is None:
    print('No match is found')
```

6.(a) How do we specify and handle absolute ,relative paths? 10

Handling Absolute and Relative Paths

Calling os.path.abspath(*path*) will return a string of the absolute path of the argument.

This is an easy way to convert a relative path into an absolute one.

```
import os
print(os.path.abspath('.'))
print(os.getcwd())
print(os.path.abspath('.\\res'))
```

Calling os.path.isabs(*path*) will return True if the argument is an absolute path and False if it is a relative path.

```
print(os.path.isabs('.'))
print(os.path.isabs(os.path.abspath('.')))
```

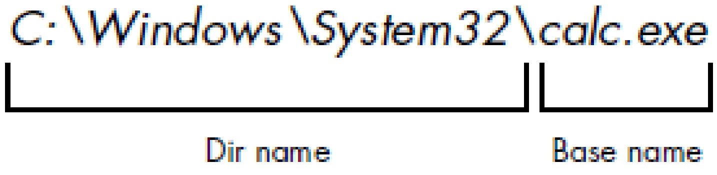
Calling *os.path.relpath(path, start)* will return a string of a relative path from the

***start* path to *path*. If *start* is not provided, the current working directory is used as the start path.**

```
import os
print(os.path.relpath('C:\\Windows', 'C:\\'))
print(os.path.relpath('C:\\Windows', 'C:\\spam\\eggs'))
```

Calling os.path.dirname(*path*) will return a string of everything that comes before the last slash in the path argument.

Calling os.path.basename(*path*) will return a string of everything that comes after the last slash in the path argument.



os.path.sep is a string containing path separators corresponding to OS

```
import os
path = 'C:\\Windows\\System32\\calc.exe'
print(os.path.basename(path))
print(os.path.dirname(path))
print(os.path.split(path))
print(os.path.sep)
print('C:\\Windows\\System32\\calc.exe'.split(os.path.sep))
```

6.(b) Explain saving of variables using shelve module. 4

We can save variables in our Python programs to binary shelf files using the shelve module.

shelve is Python object persistence utility.

```
import shelve
shelfFile = shelve.open('mydata')
cats = ['Zophie', 'Pooka', 'Simon']
shelfFile['cats'] = cats
shelfFile.close()
```

6(c) With code snippet, explain reading, extracting and creating ZIP files 6

Reading ZIP Files

To read the contents of a ZIP file, first you must create a ZipFile object by passing the name of the zipfile

```
import zipfile, os
os.chdir('D:\\Python\\Lib\\test')
exampleZip = zipfile.ZipFile('zip_cp437_header.zip')
```

```
exampleZip.namelist()

spamInfo = exampleZip.getinfo('filename_without.txt')

spamInfo.file_size

spamInfo.compress_size

print('Compressed file is %sx smaller!' % (round(spamInfo.file_size / spamInfo.compress_size, 2)))

exampleZip.close()
```

extractall() method for ZipFile objects extracts all the files and folders from a ZIP file into the current working directory.

```
import zipfile, os

os.chdir('e:\\') # move to the folder with example.zip

exampleZip = zipfile.ZipFile('demo.zip')

exampleZip.extractall()

exampleZip.close()
```

The extract() method for ZipFile objects will extract a single file from the ZIP file.

```
import zipfile, os

os.chdir('e:\\')

exampleZip = zipfile.ZipFile('demo.zip')

print(exampleZip.namelist())

exampleZip.extract("demo/Veena.txt",'d:\\')

exampleZip.close()
```

If the second argument to extract() method is not passed then the specified file will be extracted to current working directory.

If the second argument is passed and if the argument contains name of the folder which does not exist then the folder will be created and file will later extracted.

To create compressed ZIP files, we must open the ZipFile object in *write mode* by passing 'w' as the second argument.

```
import zipfile

newZip = zipfile.ZipFile('new.zip', 'w')
```

newZip.write('spam.txt', compress_type=zipfile.ZIP_DEFLATED) # deflate compression algorithm

newZip.close()

Opening the ZipFile object in *append mode* by passing 'a' as the second argument allows us to add files to existing zip files.

8.(a) Explain operator overloading with example. 7

Operator Overloading means giving extended meaning beyond their predefined operational meaning.

For example operator + is used to add two integers as well as join two strings and merge two lists. It is achievable because '+' operator is overloaded by int class and str class.

```
class Time:
   def __init__(self, hr=0, mi=0, sec=0):
       self.hr = hr
       self.mi = mi
       self.sec = sec
   def __str__(self):
       return '%.2d:%.2d:%.2d'%(self.hr,self.mi,self.sec)
   def __add__(self, other):
       sec = self.time_to_int() + other.time_to_int()
       return self.int_to_time(sec)
   def time_to_int(self):
       return self.hr*3600 + self.mi * 60 + self.sec
def int_to_time(self,sec):
       t = Time()
       mn,t.sec = divmod(sec,60)
       t.hr,t.mi = divmod(mn,60)
       return t
t1 = Time(12,40)
t2 = Time(6,30)
print(t1 + t2)
```

8(b) What are polymorphic functions? Explain with code snippet 7

Mqp1

8 (c) Illustrate the concept of inheritance with example 6

Inheritance is the ability to define a new class that is a modified version of an existing class.

When a new class inherits from an existing class, the existing class is called parent and new class is called the child.

Inheritance facilitates code reuse

```
from Deck import Deck
class Hand(Deck):
  def __init__(self, label=''):
    self.cards = []
    self.label = label
  def move_cards(self, hand, num):
    for i in range(num):
      hand.add_card(self.pop_card())
if __name__ == '__main__':
  hand = Hand('new hand')
  print(hand.cards)
  print(hand.label)
  deck = Deck()
  card = deck.pop_card()
  hand.add_card(card)
  card = deck.pop_card()
  hand.add_card(card)
  print(hand)
```