1.(a) Explain the math operators in Python from highest to lowest Precedence with an example for each. Write the steps how Python is evaluating the expression (5 - 1) * ((7 + 1) / (3 - 1)) and reduces it to a single value. (6 Marks)

Mathematical Operations in python

**** Exponent operator**

```
>>> 2 ** 10
1024
```

- **% Modulus / Remainder operator**

```
>>> 55 % 3
1
```

- **// Integer Division**

```
>>> 25 // 3
8
```

- **/ Division**

```
>>> 25 / 3
8.33
```

- *** Multiplication**

```
>>> 5 * 3
15
```

- **- Subtraction**

```
>>> 0 - 3
-3
```

- **+ Addition**

```
>>> 8 + 3
11
```

Operator Precedence

The Operator Precedence determines the **grouping of terms** in an expression and decides how an expression is evaluated.

Ex. 7 + 3 * 2 evaluates to 13 not 20

Operator	Operation	Example	Evaluates to
**	Exponent	2 ** 5	32

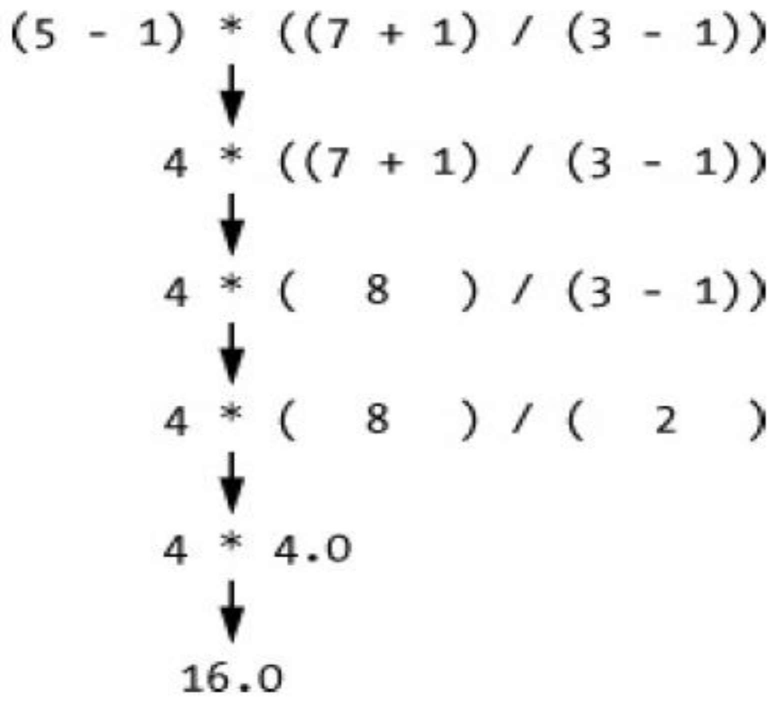
%	**Modulus**	**20 % 3**	**2**
//	**Integer Division**	**22 // 3**	**7**
/	**Division**	**5 / 4**	**1.25**
*****	**Multiplication**	**3 * 6**	**18**
-	**Subtraction**	**3 – 20**	**-17**
+	**Addition**	**8 + 19**	**27**

%, //, /, * all have same precedence which will be evaluated based on

left to right associativity

+, - all have same precedence which will be evaluated based on left to right

Associativity.



(b) Define a Python function with suitable parameters to generate prime numbers between two integer values. Write a Python program which accepts two integer values m and n (note: m>0, n>0 and m < n) as inputs and pass these values to the function. Suitable error messages should be displayed if the conditions for input values are not followed. (8 Marks

(c) Explain Local and Global Scope in Python programs. What are local and global variables? How can you force a variable in a function to refer to the global variable? (6 Marks)

Local and Global Scope

- Parameters and variables that are assigned in a called function are said to exist in that function local scope.
- Variables that are assigned outside all functions are said to exist in global scope.
- A local scope is created whenever a function is called. When the function returns, the local scope is destroyed.
- local variables can not be used in global scope.
- A variable that exists in a local scope is called a local variable, while a
- variable that exists in the global scope is called a global variable. A variable must be one or the other; it cannot be both local and global.

  // local and global variables and How can you force a variable in a function to refer to the global variable

2.(a) What are Comparison and Boolean operators? List all the Comparison and Boolean operators in Python and explain the use of these operators with suitable examples. (6 Marks)

- Comparison operators compare two values and evaluate down to a single Boolean value.

Operator	**Meaning**
==	Equal to
!=	Not Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

- Operators evaluate to True or False depending on the values you give them.

```
>>> 42 == 42            #True
>>> 42 == 99            #False
>>> 2 != 3                    #True
>>> 2 != 2                    #False
```

- The <, >, <=, and >= operators, on the other hand, work properly only with integer and floating-point values.

```
>>> 42 < 100                #True
>>> 42 > 100                #False
>>> 42 < 42                 #False
>>> eggCount = 42
>>> eggCount <= 42          #True
>>> myAge = 29
>>> myAge >= 10             #True
```

- The three Boolean operators (and, or, and not) are used to compare Boolean values.

  Boolean operators evaluate the expressions down to a Boolean value.
- The **and** and **or** operators takes two Boolean values (or expressions), so they're considered as binary operators.
- The order of preference among Boolean operators are **not, and, or**
- The **and** operator evaluates an expression to True if both Boolean values are True; otherwise, it evaluates to False.

```
>>> True and True           #True
>>>  True and False         #False
```

- The **or** operator evaluates an expression to True if either of the two Boolean values is True. If both are False, it evaluates to False.

```
>>> True or True                    #True
>>>  False or False         #False
>>> True or False           #True
```

- The not operator operates on only one Boolean value (or expression).
- The not operator simply evaluates to the opposite Boolean value.

```
>>> not True                #False
>>> not not not not True    #True
```

(b) Define a Python function with suitable parameters to generate first N Fibonacci numbers. The first two Fibonacci numbers are 0 and 1 and the Fibonacci sequence is defined as a function F as

Fn = Fn-1 + Fn-2. Write a Python program which accepts a value for N (where N >0) as input and pass this value to the function. Display suitable error message if the condition for input value is not followed. (8 Marks)

(c) What is Exception Handling? How exceptions are handled in Python? Write a Python program with exception handling code to solve divide-by-zero error situation. (6 Marks)

3.(a) What is Dictionary in Python? How is it different from List data type? Explain how a for loop can be used to traverse the keys of the Dictionary with an example. (6 Marks)

A Dictionary is a collection of many values. Indexes of Dictionary can use different data types not just integers.

Indexes in dictionary are called keys. Key along with its associated value is called key-value pair.

A dictionary is typed with braces {}.

mycat = {‘size’: ‘fat’, ‘color’: ‘gray’, ‘disposition’: ‘loud’}

Keys are ‘size’, ‘color’ and ‘disposition’. Respective values are ‘fat’, ‘gray’ and ‘loud’

mycat[‘size’] # ‘fat’

‘My cat has ’+ mycat[‘color’]+‘ fur.’ # ‘My cat has gray fur.’

Items in lists are ordered whereas items in Dictionary are unordered

The order of items matters for determining whether two lists are the same.

spam = {‘color’:‘red’, ‘age’:43}

for k in spam.keys():

print(k,sep=‘, ’) # color, age

(b) Explain the methods of List data type in Python for the following operations with suitable code snippets for each. (i) Adding values to a list ii) Removing values from a list (iii) Finding a value in a list iv) Sorting the values in a list (8 Marks)

append ()

append method adds an item to the end of the list.

Ex:- spam = [‘cat’, ‘dog’, ‘bat’]

spam.append(‘moose’)   #   [‘cat’, ‘dog’, ‘bat’, ‘moose’]

The insert() method can insert a value at any index in the list.

The first argument to insert() is the index, the second argument is the new value to be inserted.

Ex:- spam = [‘cat’, ‘dog’, ‘bat’]

spam.insert(1,‘chicken’) #   [‘cat’, ‘chicken’, ‘dog’, ‘bat’]

The return value of append() and insert() in None.

Removing values from list with remove method.

The remove() method removes the passed value from the list it is called on.

Ex:- spam = [‘cat’, ‘bat’, ‘rat’, ‘elephant’]

spam.remove(‘bat’) #   [‘cat’, ‘rat’, ‘elephant’]

Attempting to remove a value that does not exist in the list will result in a ValueError

Ex:- spam = [‘cat’, ‘bat’, ‘rat’, ‘elephant’]

spam.remove(‘chicken’) #   ValueError: list.remove(x) : x not in list

If the value appears multiple times in the list only the first instance of the value is removed.

Ex:- spam = [‘cat’, ‘bat’, ‘cat’, ‘elephant’]

spam.remove(‘cat’) #   [‘bat’, ‘cat’, ‘elephant’]

The del statement is good to use when the index of the value to remove from the list is known. The remove() method is good when we know the value to remove from the list.

Finding value in a list?

Sorting values in a list with the sort method.

Lists of number values or list of strings can be sorted with sort() method.

Ex:- spam = [2, 5, 3.14, 1, -7]

spam.sort() #   [-7,1,2,3.14,5]

spam = [‘ants’, ‘dogs’, ‘cats’]

spam.sort() #   [‘ants’, ‘cats’, ‘dogs’]

We can also pass True for the reverse keyword argument to sort in reverse order.

Ex:- spam = [2, 5, 3.14, 1, -7]

spam.sort(reverse=True) #   [5, 3.14, 2, 1, -7]

sort() method does not return any value, it sorts list in place.

(c) Write a Python program that accepts a sentence and find the number of words, digits, uppercase letters and lowercase letters. (6 Marks)

4. (a) What is the difference between copy.copy( ) and copy.deepcopy( ) functions applicable to a List or Dictionary in Python? Give suitable examples for each. (6 Marks)

copy() and deepcopy() functions

copy module has a function called copy which can be used to make a duplicate copy of a mutable list or dictionary.

import copy

spam = [‘A’, ‘B’, ‘C’, ‘D’]

cheese = copy.copy(spam)

cheese[1] = 42

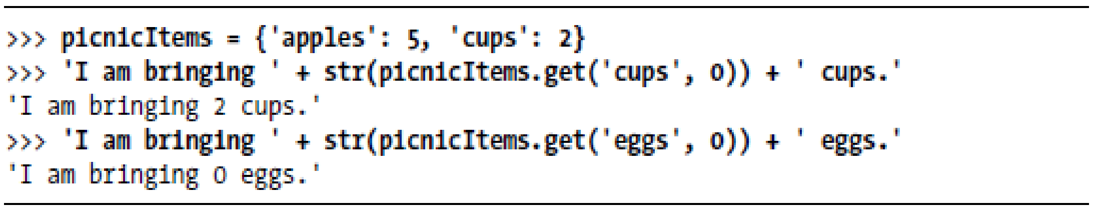
spam # [‘A’, ‘B’, ‘C’, ‘D’]

cheese # [‘A’, 42, ‘C’, ‘D’]

copy module has a function called deepcopy which can be used to make a duplicate copy of inner lists as well.

(b) Discuss the following Dictionary methods in Python with examples. (i) get() (ii) items() (iii) keys() (iv) values() (8 Marks)

get method takes two arguments, the key of the value to retrieve and a fallback value to return if the key does not exist.



```
>>> picnicItems = {'apples': 5, 'cups': 2}
>>> 'I am bringing ' + str(picnicItems.get('cups', 0)) + ' cups.'
'I am bringing 2 cups.'
>>> 'I am bringing ' + str(picnicItems.get('eggs', 0)) + ' eggs.'
'I am bringing 0 eggs.'
```

Without the get method the code would have caused error message.

keys, values and items methods

The values returned by the methods keys, values and items are not true lists, i.e. they cannot be modified and do not have an append() method. But can be used in for loops.

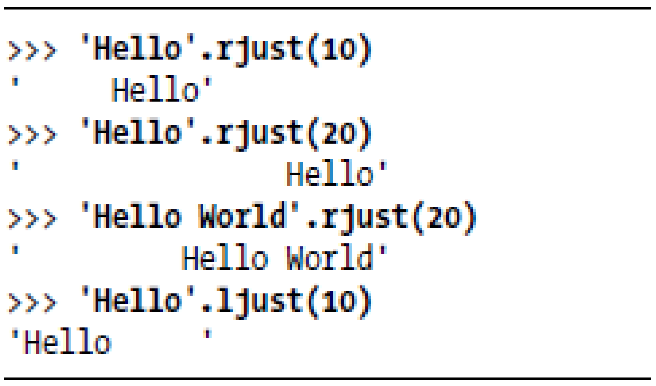
keys, values and items methods

```
spam = {‘color’:‘red’, ‘age’:43}

for v in spam.values():

    print(v,sep=‘, ’) # red, 43

for k in spam.keys():

    print(k,sep=‘, ’) # color, age

for i in spam.items():

    print(i,sep=‘, ’) # (‘color’, ‘red’), (‘age’,43)
```

(c) Explain the various string methods for the following operations with examples. (i) Removing whitespace characters from the beginning, end or both sides of a string. (ii) To right-justify, left-justify, and center a string. (6 Marks)

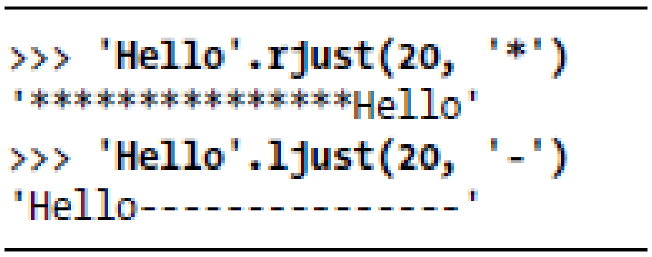
isspace() returns True if the string consists only of spaces, tabs, and newlines and is not blank.

Justifying Text with rjust(), ljust() and center() - The rjust() and ljust() string methods return a padded version of the string they are called on. The first argument is an integer length for the justified string.



```
>>> 'Hello'.rjust(10)
'     Hello'
>>> 'Hello'.rjust(20)
'               Hello'
>>> 'Hello World'.rjust(20)
'         Hello World'
>>> 'Hello'.ljust(10)
'Hello     '
```

The optional second argument to rjust() and ljust() will specify a fill character other than space character.



```
>>> 'Hello'.rjust(20, '*')
'***************Hello'
>>> 'Hello'.ljust(20, '-')
'Hello---------------'
```

Q.5 (a) Write a Python Program to find an American phone number (example: 415-555-4242) in a given string using Regular Expressions. (6 Marks)

(b) Describe the difference between Python os and os.path modules. Also, discuss the following methods of os module a) chdir() b) rmdir() c) walk() d) listdir() e) getcwd() (7 Marks)

// difference between Python os and os.path modules

The os.getcwd() function returns the current working directory.

The os.chdir() function changes the current working directory.

• Calling os.rmdir(*path*) will delete the folder at *path*. This folder must be empty of any files or folders.

os.walk() function will return three values on each iteration through the loop:

1. A string of the current folder's name (current folder, means the folder for the current iteration of the for loop.)

2. A list of strings of the folders in the current folder

3. A list of strings of the files in the current folder

Calling ***os.listdir(path)*** will return a list of filename strings for each file in the *path* argument. (Note that this function is in the os module, not os.path.)

(c) Demonstrate the copy, move, rename and delete functions ofshutil module with Python code snippet. (7 Marks)

Copying Files and Folders

shutil.copy(*source*, *destination*) will copy the file at the path *source* to the folder at the path *destination*.

If *destination* is a filename, it will be used as the new name of the copied file.

import shutil, os

os.chdir('e:\\')

shutil.copy('e:\\spam.txt', 'e:\\delicious')

shutil.copy('eggs.txt', 'e:\\delicious\\eggs2.txt')

Calling shutil.move(*source*, *destination*) will move the file or folder at the path *source* to the path *destination* and will return a string of the absolute path of the new location.

import shutil

print(shutil.move('e:\\bacon.txt', 'e:\\eggs'))

Assuming a folder named *eggs* already exists in the *C:\* directory, this shutil.move() calls says, "Move *e:\bacon.txt* into the folder *e:\eggs*."

shutil.move('e:\\bacon.txt', 'e:\\eggs\\new_bacon.txt')

Move *e:\bacon.txt* into the folder *e:\eggs*, and rename that *bacon.txt* file to *new_bacon.txt*.

If the destination does not exist then FileNotFoundError will occur.

Calling os.unlink(*path*) will delete the file at *path*.

• Calling os.rmdir(*path*) will delete the folder at *path*. This folder must be empty of any files or folders.

• Calling shutil.rmtree(*path*) will remove the folder at *path*, and all files and folders it contains will also be deleted.

import os

for filename in os.listdir(): # finds all the files in the current working directory

  if filename.endswith('.txt'):

os.unlink(filename) # deletes a file with name filename permanently

6. (a) Describe the following with suitable Python code snippet. (i) Greedy and Non Greedy Pattern Matching (ii) findall() method of Regex object. (7 Marks)

Greedy and Non-greedy Matching

Python's regular expressions are greedy by default. i.e. python will match for the longest string possible.

The Non-greedy version of the curly brackets matches the shortest string possible.

Non-greedy version has the closing curly bracket followed by a question mark ?

```
import re

regex = re.compile(r'(Ha){3,5}')

match = regex.search('HaHaHaHaHa ')

print(match.group())

regex1 = re.compile(r'(Ha){3,5}?')

match1 = regex.search('HaHaHaHaHa ')

print(match1.group())
```

findall() Method

findall() method will not return a Match object but list of strings if there are no groups in the regular expressions

If there are groups in the regular expression it returns list of tuples.

```
import re

regex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d') # has no groups

lists = regex.findall('Cell: 415-555-9999 Work: 212-555-0000')

print(lists)

regex1 = re.compile(r'(\d\d\d)-(\d\d\d)-(\d\d\d\d)') # has groups

listoftuples = regex1.findall('Cell: 415-555-9999 Work: 212-555-0000 ')

print(listoftuples)
```

(b) Explain the file Reading/Writing process with suitable Python Program. (6 Marks)

File Opening/Reading/Writing

Use open() function to open a file, which returns a file object. It takes 2 arguments, the first argument is the name of the file to be opened along with its complete path. The second argument is the **mode** in which file will be opened.

Mode – r – read, w – write, a – append

We can call read() method on file’s object.

```
helloFile = open(D:\\hello.txt')

helloContent = helloFile.read()

print(helloContent)
```

We can call readlines() method on file’s object to get a list of string values from the file, one string for each line of text.

To write into file we need to open file in **write mode (w) or append mode (a)**.

By default file will be opened in read mode.

```
baconFile = open('bacon.txt', 'w')

baconFile.write('Hello world!\n')

baconFile.close()

baconFile = open('bacon.txt', 'a')

baconFile.write('Bacon is not a vegetable.')

baconFile.close()

baconFile = open('bacon.txt')

content = baconFile.read()

baconFile.close()

print(content)
```

(c) Define assertions. What does an assert statement in python consists of? Explain how assertions can be used in traffic light simulation with Python code snippet. (7 Marks)

An *assertion* is a sanity check to make sure code isn’t doing something wrong.

An *assertion* consists of the following :

• The assert keyword

• A condition (that is, an expression that evaluates to True or False)

• A comma

• A string to display when the condition is False

```
def switchLights(stoplight):

  for key in stoplight.keys():

    if stoplight[key] == 'green':

      stoplight[key] = 'yellow'

    elif stoplight[key] == 'yellow':

      stoplight[key] = 'red'
      elif stoplight[key] == 'red':
         stoplight[key] = 'green‘
   assert 'red' in stoplight.values(), 'Neither light is red! ' + str(stoplight)
market_2nd = {'ns': 'green', 'ew': 'red'}
switchLights(market_2nd)
```

assert 'red' in stoplight.values() checks that the spotlight dictionary should contain a red value. Otherwise an error message will

be displayed.

Assertions helps to identify bugs in program in the early stages which saves a lot of future debugging efforts.

7.(a) Define classes and objects in Python. Create a class called Employee and initialize it with employee id and name. Design methods to: (i) setAge_to assign age to employee. (ii) setSalary_to assign salary to the employee. (iii) Display_to display all information of the employee. (8marks)

(b) Illustrate the concept of modifier with Python code. (5 Marks)

Modifier function modifies any of the objects passed to it as arguments.

```
class Time:
   pass
def increment(t1,sec):
   ns = t1.sec + sec
   t1.sec  = ns  % 60
   nm = t1.mi + ns //60
   t1.mi = nm % 60
   t1.hr =  t1.hr + nm//60
def print_time(t):
   print("%.2d:%.2d:%.2d"%(t.hr,t.mi,t.sec))
time1 = Time()
time1.hr = 10
time1.mi = 30
time1.sec = 50
print("Time 1 ")
print_time(time1)
```

```
print("Time 1 + 30 sec")
increment(time1,30)
print_time(time1)
```

(c) Explain init and __str__ method with an example Python Program. (7 Marks)

The init method is a special method that gets invoked when an object is instantiated.

Full name is __init__ (2 underscores followed by init followed by 2 underscores.

```
class Time:
  def __init__(self, hr=0, mi=0, sec=0):
    self.hr = hr
    self.mi = mi
    self.sec = sec
  def print_time(self):
    print('%.2d:%.2d:%.2d'%(self.hr,self.mi,self.sec))
t = Time(12,40,30)
t.print_time()
t1 = Time()
t1.print_time()
```

The __str__ method returns a string representation of an object.

```
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)
t = Time(12,40)
print(t)
```

8. (a) Define polymorphism? Demonstrate polymorphism with function to find histogram to count the number of times each letter appears in a word and in a sentence. (7 Marks)

Functions that work with several types are called polymorphic.

Polymorphism can facilitate code reuse.

```
def histogram(s):
   d = {}
   for c in s:
      if c not in d:
         d[c] = 1
      else:
         d[c] = d[c]+1
   return d
tu = ('aa','bb','aa','dd')
print(histogram(tu))
li = ['cat','bat','cat','mat','cat']
print(histogram(li))
```

(b) Illustrate the concept of pure function with Python code. (6 Marks)

Pure function does not modify any of the objects passed to it as arguments.

```
class Time:
   pass
def add_time(t1,t2):
   temp = Time()
   ns = t1.sec + t2.sec
   nm = t1.mi + t2.mi
   nh = t1.hr + t2.hr
   temp.sec = ns % 60
   temp.mi = (ns//60 + nm)%60
   temp.hr = nh + (ns//60 + nm)//60
   return temp
def print_time(t):
   print("%.2d:%.2d:%.2d"%(t.hr,t.mi,t.sec))
time1 = Time()
time1.hr = 10
time1.mi = 30
```

```
time1.sec = 50

print("Time 1 ")

print_time(time1)

time2 = Time()

time2.hr = 11

time2.mi = 30

time2.sec = 1

print("Time 2 ")

print_time(time2)

print("Time 1 + Time 2 ")

xx =  add_time(time1,time2)

print_time(xx)
```

(c) Define Class Diagram. Discuss the need for representing class relationships using Class Diagram with suitable example. (7 Marks)

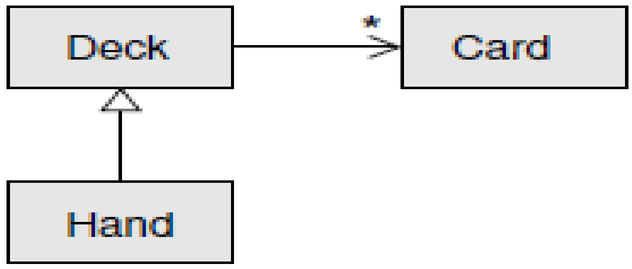
Class Diagrams

Objects in one class might contain references to objects in another class. For example, each Rectangle contains a reference to a Point, and each Deck contains references to many Cards. This kind of relationship is called **HAS-A**, as in, “a Rectangle has a Point.”

One class might inherit from another. This relationship is called IS-A, as in, “a Hand is a kind of a Deck.”

One class might depend on another in the sense that objects in one class take objects in the second class as parameters, or use objects in the second class as part of a computation. This kind of relationship is called a dependency.

H
A
S
A
A


Deck
*
Card
Hand