Explain Regular expressions in python

- A regular expression in a programming language is a special text string used for describing a search pattern
- It is extremely useful for extracting information from text such as code,files,log,spreadsheets or even documents.
- While using a regular expression the first thing to recognize is that everything is essentially a character, and we are writing patterns to match a specific sequence of characters reffered to as string.
- For instance a regular expression could tell a program to search for specific text from the string and then to printout the result accordingly.An expression can include-
- Text matching
- Repetition
- Branching
- Pattern-Composition
- In python, a regular expression is denoted as RE(regexes or regex pattern) and is imported through the **re module**
- Python supports regular expressions through libraries

❖Explain Match Function and give one example.

Re Match Function: This function attempts to match RE pattern to string with

```
import re
re.match (pattern, string or character,flags=0(optional))
```

optional flags

Here is the description of the parameters

Sr.No.	Parameter & Description
1	pattern This is the regular expression to be matched.
2	string This is the string, which would be searched to match the pattern at the beginning of the string.
3	flags(optional) You can specify different flags using bitwise OR (|). These are modifiers, which are listed in the table below.

The re.match function returns a match object on success, none on failure. We use group(num) or groups() function of match object to get matched expression.

Sr.No.	Match Object Method & Description
1	group(num=0) This method returns entire match (or specific subgroup num)
2	groups() This method returns all matching subgroups in a tuple (empty if there weren't any)

Example:

```
import re
list=["Apple Fruit","Apple Cell Phone","Car Vehicle","Argentina Country","Hockey Game"]
for element in list:
   ans=re.match("(A\w+)\W(C\w+)",element)
   if ans:
      print(ans.groups())
```

Output:

('Apple', 'Cell')

('Argentina', 'Country')

In this given example, the match function is used to match RE Pattern to string with optional flags.

In this method,the expression “w+” and “\W” will match words starting with letter ‘A’ and thereafter,anything which is not started with ‘A’ is not identified.

To check match for each element in the list or string, we run the **for loop.**

- ❖ What is multithreading?Write a short note on thread module
  - A thread is the smallest unit of a program or process executed independently or scheduled by the Operating System.
  - In the computer system, an Operating System achieves multitasking by dividing the process into threads.
  - A thread is a lightweight process that ensures the execution of the process separately on the system.
  - Multithreading refers to concurrently executing multiple threads by rapidly switching the control of the CPU between threads.
  - A thread has a beginning, an execution sequence, and a conclusion. It has an instruction pointer that keeps track of where within its context it is currently running.
    1. It can be pre-empted (interrupted)
    2. It can temporarily be put on hold (also known as sleeping) while other threads are running - this is called yielding.

Short note on thread module:

- **There are two modules that support the use of threads in python 3:**
  1. **_thread**
  2. **threading**
- **The “thread” module has been “deprecated” for quite a long time. Hence in python 3, the module “thread” is not available to**

use anymore. However it is renamed to "_thread" for backward compatibilities in python 3

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

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

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

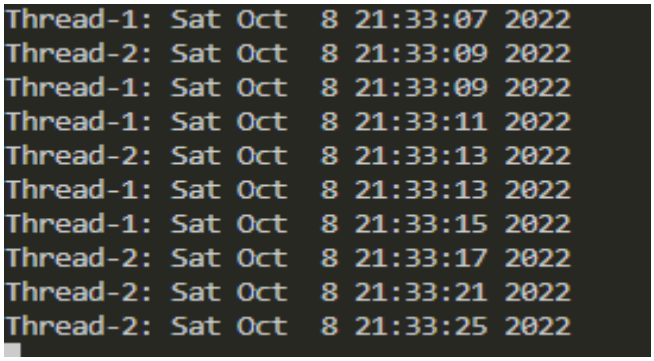
```
import _thread
import time

#Define a function for the thread
def print_time( threadName, delay):
   count = 0
   while count < 5:
      time.sleep(delay)
      count += 1
      print("%s: %s" % ( threadName, time.ctime(time.time())))

#Create two threads as follows
try:
   _thread.start_new_thread( print_time, ("Thread-1", 2, ) )
   _thread.start_new_thread( print_time, ("Thread-2", 4, ) )
except:
   print("Error: unable to start thread")

while 1:
   pass
```

Output:



Although it is very effective for low-level threading, but the *thread* module is very limited compared to the newer threading module.

- Explain Random Module with Any 5 functions:

- You can generate random numbers in Python by using random module.
- Python offers random module that can generate random numbers.
- The random module gives access to various useful functions and one of them Is being able to generate random numbers
- Random numbers are used for games, simulations, testing, security, and privacy applications.
- Functions of random module include:

1. choice(seq)

   This Function returns a random item from a list,tuple ot string

- **seq** − This could be a list, tuple, or string.

2. randrange ([start,] stop [,step])

   This function returns a randomly selected element from range(start, stop, step).

- **start** − Start point of the range. This would be included in the range.

- **stop** − Stop point of the range. This would be excluded from the range.
- **step** − Steps to be added in a number to decide a random number.

3. random ( )
   This method returns a random float r, such that 0 is less than or equal to r and r is less than 1.
4. shuffle (lst)
   shuffle() randomizes the items of a list in place
   lst − This could be a list or tuple
5. uniform(x, y)
   uniform() returns a random float r, such that x is less than or equal to r and r is less than y.

   x − Sets the lower limit of the random float.

   y − Sets the upper limit of the random float

Explain Time module with any 5 functions:

Python has defined a module,"time" which allows us to handle various operations regarding time,its conversions and representations,which finds its use in various applications of life

The beginning of time started measuring from jan1 1970 and thiss very time is termed "epoch" in python

Functions of time module:

1.The `time()` function returns the number of seconds passed since epoch.

```
import time
seconds = time.time()
print("Seconds since epoch =", seconds)
```

2.The `ctime()` function takes seconds passed since epoch as an argument and returns a string representing local time.If no argument is passed the time is calculated till present

```
import time

# seconds passed since epoch
seconds = 1545925769.9618232
local_time = time.ctime(seconds)
print("Local time:", local_time)
```

3. The `sleep(sec)` function suspends (delays) execution of the current thread for the given number of seconds.

```
import time

print("This is printed immediately.")
time.sleep(2.4)
print("This is printed after 2.4 seconds.")
```

4. gmtime(sec):This function returns a structure with 9 values each representing a time attribute in sequence.It converts seconds into time attributes(days,years,months)till specified seconds from epoch

5.asctime(“time”): This function takesa time attributed string produced by gmtime() and returns a 24 character string denoting time

What are modules?How to use?what areAdvantages of Modules?

- 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.
- Importing a module

  The “import” statement
- You can use any Python source file as a module by executing an import statement in some other Python source file. The *import* has the following syntax –
- import module1[, module2[,... moduleN]
- When the interpreter encounters an import statement, it imports the module if the module is present in the search path.
- A search path is a list of directories that the interpreter searches before importing a module.
- For example, to import the module math, you need to put the following command at the top of the script −

```
import math
math.pi
```

A module is loaded only once,regardless of the number of times it is imported

This prevents the module execution from happening repeatedly, if multiple imports occur

The from import Statement

Python’s from statement lets you import specific attributes from a module without importing the module as a whole.

from math import sqrt, factorial

```
print(sqrt(16))
print(factorial(6))
```

if we simply do "import math", then math.sqrt(16) and math.factorial() are required.

From import * Statement

The * symbol used with the from import statement is used to import all the names from a module to a current namespace.

Syntax:from module_name import *

```
from math import *
print(sqrt(16))
print(factorial(6))
```

if we simply do "import math", then math.sqrt(16) and math.factorial() are required.

Advantages of modules –

- **Reusability** : Working with modules makes the code reusable.
- **Simplicity:** Module focuses on a small proportion of the problem, rather than focusing on the entire problem.
- **Scoping:** A separate namespace is defined by a module that helps to avoid collisions between identifiers.

What is OOP? Explain the concept of OOPs

In Python, object-oriented Programming (OOPs) is a programming paradigm that uses objects and classes in programming. It aims to implement real-world entities like inheritance, polymorphisms, encapsulation, etc. in the programming. The main concept of OOPs is to bind the data and the functions that work on that together as a single unit so that no other part of the code can access this data.

Main Concepts of Object-Oriented Programming (OOPs)

```
class ClassName:

    # Statement-1

    .

    .

    .

    # Statement-N
```

- Class
- Objects
- Polymorphism
- Encapsulation
- Inheritance
- Data Abstraction

Class

A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods.

Creating a class

```
Class MyClass:
    i=12345
    def f(self):
        return”hello world”
```

The object is an entity that has a state and behavior associated with it. It may be any real-world object like a mouse, keyboard, chair, table, pen, etc. Integers, strings, floating-point numbers, even arrays, and dictionaries, are all objects

For above class object can be created using a statement like

```
X=MyClass()
```

Inheritance

Inheritance is the feature using which an object of one class inherits the features of another class from which it is derived

The primary advantage of inheritance is that the derived class can access the data and the functions of the base class.

This helps in implementing the reusability of the code.

The class which is inherited is called the base class and the class which is derived from the base class is called the derived class.

Syntax for creating a derived class:

Class derived-class(base class):

<Body of the class>

Polymorphism

Polymorphism simply means having many forms. For example, we need to determine if the given species of

birds fly or not, using polymorphism we can do this using a single function.

Polymorphism helps an operator and function behave differently in different classes

This feature has largely contributed in the versatility of object oriented programs.

Features like operator overloading and function overloading reduces the complexity of the program.

Data Abstraction

Data abstraction refers to representing only the essential features, without including an background details.

Also, when we do not want to give out sensitive parts of our code implementation and this is where data abstraction is used.

Encapsulation

Encapsulation is one of the fundamental concepts in object-oriented programming (OOP).

It describes the idea of wrapping data and the methods that work on data within one unit.

This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data.

A class is an example of encapsulation as it encapsulates all the data that is member functions, variables, etc.

Explain Constructor

Constructors are generally used for instantiating an object. The task of constructors is to initialize(assign values) to the data members of the class when an object of the class is created. In Python the __init__() method is called the constructor and is always called when an object is created.

Syntax of constructor declaration :

```
def __init__(self):
    # body of the constructor
```

Types of constructors :

default constructor: The default constructor is a simple constructor which doesn't accept any arguments. Its

definition has only one argument which is a reference to the instance being constructed.

parameterized constructor: constructor with parameters is known as parameterized constructor. The parameterized constructor takes its first argument as a reference to the instance being constructed known as self and the rest of the arguments are provided by the programmer.

Write a Program to Demonstrate the use of inheritance

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

#Use the Person class to create an object, and then execute the printname method:

x = Person("John", "Doe")
x.printname()
```

Write a Program to Demonstrate the use of multiple inheritance

```
# Multiple Inheritance using two classes:


class Father():
    def Driving(self):
        print("Father Enjoys Driving")
class Mother():
    def Cooking(self):
        print("Mother Enjoys Cooking")
class Child(Father, Mother):
    def Playing(self):
        print("Child Loves Playing")
c = Child()
```

c.Driving()

c.Cooking()

c.Playing()

Here, the father and Mother are the Base classes where we have two print statements and a child class that contains all the methods of the father and mother class.

The child class is also known as Derived class.

We are creating the object for the child class through which we can access the functions of father, mother, and child.