OOPs In Python

Q What is object oriented programming?

⇒ Object-oriented programming is about creating objects that contain both data and functions.

Q What is the structure of object-oriented prog?

⇒ The structure of object-oriented prog include the following.

- Classes: are the undefined data types that act as the blueprint for individual objects. attributes and methods.
- Methods: are functions that are defined inside a class that describe the behaviours of an object
- Objects: are instances of a class created with specifically defined data. Objects can correspond to real world objects or an abstract entity.
- Attributes: are defined in the class template and represent the state of an object. Objects will have data stored in the attributes field.

Q What is Class? 1

⇒ A class is a blueprint or set of instructions to build a specific type of object. It is a basis concept of Object oriented-Programming which revolve around the real-life entities.

Syntax of Class in Python :

```
class Car:                          → In PascalCase.
    color = "blue"
    model = "sport"                 # data
    def calculate_avg_speed(km, time):   → In snake_case.
        # some code.
```

1.1 Difference between Structure & Class ?

⇒ Class	Structure
① It is declared using class keyword.	① It is declared using struct keyword.
② Members of class are private by default.	② Members of structure are public by default.
③ Occupies more space	③ Occupies less space.
④ Object for a class is created in the heap memory.	④ Object for a structure is created in the stack memory.
⑤ Object is created using "new" keyword.	⑤ May or may not use "new" keyword while crating object.

1.2 Similarities between Structure & Class.

① ~~Structures are value types and classes are reference types.~~

① Both can have constructors, methods, properties fields, constants, enumerations, events.

② Structures and classes can implement interface.

③ Both of them can have constructors with and without parameter.

1.3 Access Modifiers: Access modifiers play an important role to protect the data from unauthorized access as well as protecting it from getting manipulated. The access modifies define how the members of the class can be accessed.

There types of access modifiers are—

(i) Public.

(ii) Protected.

(iii) Private

- Public: are declared publically & easily accessible from any part of the program. All data members and functions are public by default.
- Private: member are declared private, are accessible within the class only private access modifier is the most secure access modifier. Declared by adding a double underscore '__' symbol before.

properties / data member.
functionality / member function.

Protected: Only accessible to a class derived from it. Data members of a class are declared protected by adding a single underscore '_' before the data members.

Syntax:

```
class Lang:
    firstlang = "c"
    _secondlang = "python"
    __thirdlang = "java"
print(Lang.firstlang)
print(Lang._secondlang)
print(Lang._Lang__thirdlang)  # using name mangling we can access private modifier.
```

1.4 Member Function: Functions which define the behaviour of an object of that class are called member functions.

Inner class: Class inside a class.

1.5 Constructors: The task of constructors is to initialize (assign value) to the data members of the class when an object of the class is created. In python __init__() method is called

the constructor and is called when an object is created.

Syntax:

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

Default Constructor: Is a simple constructor which does not accept any argument. Its definition has only one argument which is a reference to the instance being constructed.

Parameterized Constructor: Its first argument as a reference to the instance being constructed known as self and the rest of the arguements are provided by the programmer.

- Whenever we created an object of any class the default constructor is getting called by ~~itself~~ itself.
- We can use the built-in @classmethod decorator to write multiple constructors.

Destructor: Destructor is a member method of a class. It deletes the memory of the object.
[--del--()]

[2] What is an Object?

⇒ An object is an instance of a class. A class is like a blueprint while an instance is a copy of the class with actual values. Python is object-oriented programming language that stresses on objects. i.e. It mainly emphasizes functions.

Syntax:

```
Obj = Myclass()
print(obj.x)
```

[2.1] Difference between Object and class.

Object	Class
(I) Object is an instance of a class.	(I) Class is a blueprint or template from which objects are created.
(II) Objects are real world entity.	(II) Class is a group of similar objects.
(III) It is physical entity.	(III) logical entity.
(IV) Everything in python is an object.	(IV) Declared using class key-word. eg⇒ class Student()
(V) Objects allocates memory when it is created.	(V) Class doesn't allocated memory when it is created.

[3] Methods: The purpose of a method is to process the variables provided in a class or in the method. Type of Methods -

- Instance methods.
  - (a) Accessor methods.
  - (b) Mutator methods.
- Class methods.
- Static methods.

[3.1] Instance Methods: Instance methods are the methods which act upon the instance variables of the class. Intance methods are bound to instance (or object) and hence called as: instancename.method(). While calling the instance methods, we need not pass any value to the 'self' variable.

[3.1.1] Accessor Methods: It simply access or read data of the variables. They do not modify the data in the variables. It also called as getter methods.

```
def getName(self):
    return self.name
```

[3.1.2] Mutator Methods: Are the methods which not only read the data but also modify them. They also called as setter methods.

```
def setName(self, name):
    self.name = name
```

3.2 Class Method: Class methods are the methods which act on the class variables static variables. These methods are written using @classmethod decorator above them.

3.3 Static Method: Static method are used when some processing is related to the class but does not need the class or its instances to perform any work. Static methods are written with a decorator @steticmethod above them.

4 Features of OOPs:

4.1 Polymorphism: The word polymorphism means having means having many forms. In programming polymorphism means the same functions name but differentseres being used for differents types.

Syntax:

```
#len() being used for a string.
print(len("sowrav")) #6
#len() being used for a list.
print(len([10,20,30])) #3
```

4.1.1 Functions Overloading:- Functions overloading is the process when the same functions can be used multiple times by passing a different number of parameters as arguements. But python does not support functions overloading. An error gets thrown if we implement the function overloading.

But does not mean there is no other way to implement this. We will define functions by setting one or more parameters as None.

```
class name:
    def fun(self, p1=None, p2=None,.....)
```

4.1.2 Dunder or Magic Methods:- Dunder methods in python are the methods having two prefix and suffix underscores in the method name. Dunder here means "Duble Under". Few examples are __init__, __add__, __len__, __repr__,

4.1.3 Operator Overloading Polymorphism:- If any operator performs additional actions other than what it is meant for, it is called operator overloading. Operator overloading is an example for polymorphism

Operators & Corresponding Magic Methods:

+	object.__add__(self, other)	/=	object.__idiv__(self, other)
-	object.__sub__(self, other)	//=	object.__ifloordiv__(self, other)
*	object.__mul__(self, other)	%=	object.__imod__(self, other[, modulo])
/	object.__div__(self, other)	**=	object.__ipow__(self, other)
//	object.__floordiv__(self, other)	<	object.__lt__(self, other)
%	object.__mod__(self, other)	<=	object.__le__(self, other)
**	object.__pow__(self, other[, modulo])	>	object.__gt__(self, other)
+=	object.__iadd__(self, other)	>=	object.__ge__(self, other)
-=	object.__isub__(self, other)	==	object.__eq__(self, other)
*=	object.__imul__(self, other)	!=	object.__ne__(self, other)

4.1.4 Method Overloading Polymorphism: If a method is written such that it can perform more than one task. it is called method overloading.

4.1.5 Method Overriding Polymorphism: When there is a method in super class writing the same method in the sub class so that it replaces the super class method is called method overriding.

4.2 Inheritance: Deriving new classes from the existing classes such that the new classes inherit all the members of existing classes, is called inheritance.

Syntax

```
class Baseclass:
    {Body}
class DeriveClass (BaseClass):
    {Body}
```

4.2.1 Single Inheritance: Deriving one or more sub classes from a single base is called single inheritance. In single inheritance we always have only one base class, but there ~~class from~~ can be n number of sub classes derived from it.

4.2.2 Multiple Inheritance: Deriving sub classes from multiple base classes is called multiple inheritance. There will be more than one super class and there may be one or more sub classes.

4.2.3 The super() Method: super() is a built-in method which is useful to call the super ~~call~~ class constructor or methods from the sub class. Any constructor written in the super class is not available to the sub class if the sub class has a constructor. By calling a ~~constructor~~ the

super() class constructor using the super method from inside the sub class constructor. super() is a built-in method in Python that contain the history of super class method.
super() can use as:

```
super().__init__() #call super class constructor
super().__init__(argument) #pass argument
super().method() #call super class method.
```

4.2.4 Difference between Inheritance & polymorphism.

Inheritance	Polymorphism
(I) Applied to class	(I) Applied to methods
(II) It support the concept of reusability and reuse reduces code length in OOPs.	(II) It allows the objects to decide which form of the function to implement at compile time as well as run time.
(III) It can be single, hybrid, multiple & hierarchical and multilevel inheritance.	(III) Whereas it can be compiled-time polymorphism (overload) as well as run-time polymorphism (overriding).

4.3 Encapsulation: 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.

4.3.1 Advantages of Encapsulation: The main advantages of using encapsulation is the security of Data. It protects an objects from unauthorized access. Other advantages are Data Hiding, Simplicity and aesthetics.

Class member access specifier	Access from own class	Accessible from derived class	Accessible from object
Private	Yes.	No.	No.
Protected	Yes.	Yes.	No.
Public	Yes.	Yes.	Yes.

4.4 Abstraction:

In python, an abstraction is used to hide the irrelevant data/class in order to reduce the complexity. It also enhances the application efficiency.

Python provides the **abc** module to use the abstraction in the python program.

Syntax:

```
from abc import ABC
class ClassName(ABC):
```

4.4.1 Difference between Abstraction Vs Encapsulation

Abstraction.	Encapsulation.
(i) Abstraction works on the design level.	(i) Encapsulation works the application level.
(ii) It is implemented to hide the unnecessary data.	(ii) It is the mechanism of hiding of hiding the code and the data.
(iii) It highlights what is the work of object instead of how the object works.	(iv) It focuses on the inner details of how the objects is works.

5 Misc Question on OOPs in Python

5.1 Difference between Procedural & Objected Oriented programming.

Procedural Programming	Object-Oriented Programming
① It is less secure than OOPs.	① Data hiding is possible in Object oriented programming due to abbraction.
② It follows top down approach	② It follows bottom-up approach
③ It is structure-oriented	③ It is object-oriented.
④ There are no access modifiers.	④ Access modifiers in OOPs are private, public & protected.
⑤ Not appropriate for complex problem	⑤ Appropriate for complex problem.
⑥ Example: C, Pascal	⑥ Example: Python, Java.

5.2 What kind of a programming language is Python?

→ Python is basically a High-Level Object-Oriented

programming language. It also an interpreted language i.e. it uses line-by-line execution.

5.3 What is suite in Python?

⇒ suite is either a simple statement in one line or it is a newline followed by an indented block which consists of statements.

5.4 What is 'init' keyword in python.

⇒ The __init__.py file lets the python interpreter know that a directory contains code for a python module. It can be blank. Without one, you cannot import modules from another folder.

The __init method is similar to constructors in C++ & Java. Constructors are used to initilize the object's state

5.5 Difference between Modules & packages in python.

⇒ The module is simple Python file that contains collections of functions and global variables and with having a .py extension file.

The package is a simple directory having

collection of modules. The dictory contains Python modules and also having __init__.py file by which the interpreter interprets it as a Package.

5.6 What is 'self' keyword in python?

⇒ The 'self' parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

5.7 Difference between Pickling and Unpickling?

(#) In python the pickle module accepts any python object, transforms it into a string representation and dumps it into a file by using the dump function. This process known as pickling. The function used for this process is pickle.dump(). Also called serialization

(#) The process of retrieving the original python object from the stored string representation is called unpickling. The function used for this process is pickle.load(). Also called de-serialization.

5.8 What does *args and **kwargs mean?

⇒ When you are not clear how many arguments you need to pass to a particular functions, then we use *args and **kwargs.

The *args keyword represents a varied number of arguments. It is used to add together the values of multiple arguments.

The **kwargs keyword represents an arbitrary number of arguments that are passed to a functions. **kwargs keywords are stored in a dictionary.

5.9 What is "open" & "with" statement in python.

⇒ Both statements are used in case of file handling.

5.10 What is Pythonpath?

⇒ PYTHONPATH is an environment variable which you can set to add additional directories where python will look for modules and packages.

5.11 What is .pyc file.

⇒ .pyc contains the bytecode of your program.

5.12 Shallow copy & Deep copy:

⇒ Shallow copies duplicate as little as possible. It is creating a new object and then copying the non static fields of the current object to the new object

Deep copy duplicate everything. It is creating a new object and then copying the non static fields of the current object to the new object.