29-01-2026
Thursday

OOPS THEORY Interview Questions

Q1, Q2, Q7, Q8, Q9, Q10, Q11, Q12, Q13, Q14, Q5, Q23, Q24, Q25, Q30

Questions

(Q1) OOPS?

(Ans1) Object oriented programming
→ Software with bunch of objects talking to each other.
→ object (collection of data & methods)

(Q2) Why OOPs?

(Ans-2) ① Code looks like real world object, so developer and users understand better.
② Due to encapsulation we can change code without affecting how it's used.
③ Suitable for large companies as code is well organised.

(Q7) Class?

→ Blueprint used to create objects.
→ Contains :- ⓐ Variables.
ⓑ Methods.

(Q8) Object?

→ instance of a class.
→ Used to access data & method of class.
→ represents real world entity with data & methods.

(Q9) OOPs Features

→ Encapsulation :- Binding method & data together & protect data.

→ Abstraction :- Show required detail & hide internal detail.

→ Inheritance :- One class acquires property of others.

→ Polymorphism :- Same function behaves differently in different situations.

(Q10) Encapsulation

→ Wrapping data & methods together inside a class.

→ Protects data by allowing access only through methods. (data hide)

(Q11) Abs

→ Sho
hide a
user's
→ Imp

(Q12) I

→ Ch
p
Imp →
→
[child
pro
cal

(Q13)

→ San
diff
Type
(2) Comp

(Q11) Abstraction ?

→ Shows only **neccessary** details & **hide** any **unnecessary** details from the user's.

→ Implemented using **classes & variables**.

(Q12) Inheritence & Purpose ?

→ **Child** class derives property from **parent** class.

Imp
→ Code **Reusability**

→ Runtime Polymorphism (**method overriding**)

[child class provide its **own** method already provided in parent class, and method call is decided at **runtime**].

Eg 1 - a = Dog()
a.sound()

(Q13) Polymorphism & Types

→ **Same function** or method works **differently** in different situations.

Typs
① Runtime Polymorphism (**method overriding**)

② Compiletime Polymorphism (method overloading)

(Q14) access specifier & significance in oops

→ Keywords that controls visibility & accessibility of class members (data & methods).

→ Types :-

ⓐ Public :- accessible from anywhere.
ⓑ Protected :- " " class & subclass.
ⓒ Private :- " " only inside class.

→ Helps achieve encapsulation & data hiding in OOPs.

(Q15) Overloading and Overriding

→ Overloading :- Same method name with different parameters in the same class.
→ Compile time (decided)

→ Overriding :- Same method name in parent & child class with different implementation.
→ Runtime (decided)

(Q23) Constr
→ A spec
object
created
→ Python

(Q24) Type
① Paramet
→ No
→ def — in
② Non Pa
→ Const
(ex
→ def

(Q25) dest
→ Calle
→ Used
Eg :- class
de

(Q23) Constructor ?

→ A special method used to initialise object data when object is created.

→ Python! - `__init__(self):`

(Q24) Types of constructors in Python

① Parameterized

→ We can input (argument)

→ `def __init__(self, name):`

② Non Parameterized

→ Constructor without parameter. (except (self).

→ `def __init__(self):`

(Q25) destructor in Python.

→ Called when object is 'deleted'.

→ Used to release resources (Files).

Eg:-
```
class Test:
    def __del__(self):
        del t    (t = Test())
```

(Q 30) Abstract class in Python

→ Class that cannot be instantiated. (eg:- class Animal)
(a = Animal() X wrong, it's meant to used by other class like class DOG)

→ Used only for inheritance

→ Forces child class to implement required methods.

→ Created as below

```
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def sound(self):
        pass

class DOG(Animal):
    def sound(self):
        print("Bark")
```