Top 20+ OOP Questions and Answers in Python

1. What is OOP?

OOP (Object-Oriented Programming) is a programming paradigm based on the concept of "objects" that contain data and methods. Key principles are: Encapsulation, Inheritance, Polymorphism, Abstraction.

2. Define Encapsulation with examples.

Encapsulation hides internal object details and only exposes what's necessary.

Example:

```
class Person:
    def __init__(self, name):
        self.__name = name

    def get_name(self):
        return self.__name
```

3. How does Inheritance work in Python?

Inheritance lets a class inherit methods and properties from a parent class.

Example:

```
class Animal:
    def speak(self): return "Sound"

class Dog(Animal):
    def speak(self): return "Bark"
```

4. Explain Polymorphism with code.

Polymorphism allows different classes to have methods with the same name.

Example:

```
class Cat:
    def sound(self): return "Meow"


class Dog:
    def sound(self): return "Bark"
```

5. Difference between classmethod and staticmethod?

- classmethod takes cls as the first argument.
- staticmethod takes no default arguments.

Example:

```
@classmethod
@staticmethod
```

6. What is the use of __str__()?

__str__() defines how an object is printed (string representation).

Example:

```
class Person:
    def __str__(self): return "Person object"
```

7. Private vs Protected in Python?

- Protected: _var (convention)
- Private: __var (name mangled)

8. What is MRO (Method Resolution Order)?

MRO defines the order in which classes are searched when executing a method.

Use: print(ClassName.__mro__)

9. Can Python support multiple inheritance?

Yes. Example:

```
class A: pass

class B: pass

class C(A, B): pass
```

10. Explain abstraction and how it's achieved in Python.

Abstraction hides implementation using abstract classes and methods.

Example:

```
from abc import ABC, abstractmethod

class Shape(ABC):

    @abstractmethod

def area(self): pass
```

11. How does constructor work?

__init__() method is automatically called when an object is created.

12. Difference between overloading and overriding?

- Overloading: same method name, different params (not directly supported in Python).

- Overriding: subclass modifies parent class method.

13. Explain duck typing.

"If it looks like a duck and quacks like a duck, it's a duck."

Example:

```
def add(a, b): return a + b
```

14. What are magic methods?

Special methods with double underscores. Example: __init__, __str__, __add__

15. How are exceptions handled in OOP design?

Using try-except blocks and custom exceptions.

Example:

```
try:
    # code
except ValueError:
    # handle
```

16. Difference between composition and inheritance?

- Inheritance: "is-a"
- Composition: "has-a"

Example:

```
class Car:
    def __init__(self): self.engine = Engine()
```

17. What is object slicing?

Refers to losing parts of a derived class when assigned to base class (doesn't occur in Python).

18. Can you override a static method?

Yes, you can override it in a subclass like any other method.

19. How to implement singleton?

```
class Singleton:
    _instance = None
    def __new__(cls):
        if not cls._instance:
            cls._instance = super().__new__(cls)
        return cls._instance
```

20. How to protect attributes from modification?

Use private variables and read-only properties.

Example:

```
class Test:
    def __init__(self): self.__value = 10
    @property
def value(self): return self.__value
```