Mastering Object-Oriented Programming in Python

Comprehensive Q&A Guide on Classes and OOP Concepts

Object-Oriented Programming and Classes in Python

https://www.linkedin.com/in/mohd-maaz-08691a366/

1. Question: What is a class in Python?

Answer: A class is a blueprint for creating objects that defines attributes and methods.

Code Example:

```python
class Person:
    pass


p = Person()
print(type(p))
```

2. Question: How do you create an object from a class?

Answer: By calling the class name like a function.

Code Example:

```python
class Car:
    pass

my_car = Car()
print(my_car)
```

3. Question: What does self refer to in a class method?

Answer: self refers to the current instance of the class.

Code Example:

```python
class Dog:
    def bark(self):
        print("Woof!")
```

```
d = Dog()
d.bark()
```

4. Question: What is the init method?

Answer: It is the constructor method automatically called when an object is created.

Code Example:

```python
class Student:
    def __init__(self, name):
        self.name = name

s = Student("Alice")
print(s.name)
```

5. Question: Can a class exist without init?

Answer: Yes, Python provides a default constructor if init is not defined.

Code Example:

```python
class Empty:
    pass

obj = Empty()
obj.value = 42
print(obj.value)
```

6. Question: How do you add attributes dynamically to an object?

Answer: By direct assignment after object creation.

Code Example:

```python
class Book:
    pass
```

```
b = Book()
b.title = "Python OOP"
b.pages = 300
print(b.title, b.pages)
```

7. Question: What are instance attributes?

Answer: Attributes unique to each object instance.

Code Example:

```python
class Cat:
    def __init__(self, color):
        self.color = color

c1 = Cat("black")
c2 = Cat("white")
print(c1.color, c2.color)
```

8. Question: What are class attributes?

Answer: Attributes shared by all instances of the class.

Code Example:

```python
class Dog:
    species = "Canis familiaris"


print(Dog.species)
d = Dog()
print(d.species)
```

9. Question: How do you define a class method?

Answer: Using the classmethod decorator with cls as first parameter.

Code Example:

```python
class Circle:
    pi = 3.14159
```

```
    @classmethod
    def get_pi(cls):
        return cls.pi

print(Circle.get_pi())
```

10. Question: How do you define a static method?

Answer: Using the staticmethod decorator with no self or cls.

Code Example:

```python
class Math:
    @staticmethod
    def multiply(x, y):
        return x * y

print(Math.multiply(4, 5))
```

11. Question: What is inheritance in Python?

Answer: A class can inherit attributes and methods from another class.

Code Example:

```python
class Animal:
    def speak(self):
        print("Some sound")

class Dog(Animal):
    pass

d = Dog()
d.speak()
```

12. Question: How do you call a parent class method?

Answer: Using super().

Code Example:

```python
```

```
class Parent:
    def show(self):
        print("Parent")

class Child(Parent):
    def show(self):
        super().show()
        print("Child")

Child().show()
```

13. Question: What is method overriding?

Answer: Redefining a method in child class with the same name.

Code Example:

```python
class Bird:
    def fly(self):
        print("Flying")
```

```
class Ostrich(Bird):
    def fly(self):
        print("Cannot fly")

Ostrich().fly()
```

14. Question: What is multiple inheritance?

Answer: Inheriting from more than one parent class.

Code Example:

```python
class Flyer:
    def fly(self):
        print("Flying")

class Swimmer:
    def swim(self):
        print("Swimming")
```

```
class Duck(Flyer, Swimmer):
    pass

Duck().fly()
Duck().swim()
```

15. Question: What is polymorphism?

Answer: Using the same interface for different underlying types.

Code Example:

```python
class Cat:
    def sound(self):
        print("Meow")

class Dog:
    def sound(self):
        print("Woof")
```

```
for pet in [Cat(), Dog()]:
    pet.sound()
```

16. Question: How do you make an attribute private?

Answer: Prefix with double underscore to trigger name mangling.

Code Example:

```python
class Account:
    def __init__(self):
        self.__balance = 1000

    def get_balance(self):
        return self.__balance

a = Account()
print(a.get_balance())
```

17. Question: What is a property decorator?

Answer: It lets you define methods that behave like attributes.

Code Example:

```python
class Person:
    def __init__(self):
        self._name = ""

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        self._name = value

p = Person()
p.name = "Bob"
```

```
print(p.name)
```

18. Question: How do you implement str?

Answer: Override the str method to return a readable string.

Code Example:

```python
class Book:
    def __init__(self, title):
        self.title = title

    def __str__(self):
        return f"Book: {self.title}"

print(Book("Python"))
```

19. Question: How do you overload the + operator?

Answer: By implementing the add method.

Code Example:

```python
class Number:
    def __init__(self, value):
        self.value = value

    def __add__(self, other):
        return Number(self.value + other.value)

n1 = Number(10)
n2 = Number(20)
result = n1 + n2
print(result.value)
```

20. Question: What is super() used for?

Answer: To call methods from the parent class.

Code Example:

```python
```

```
class A:
    def greet(self):
        print("Hi from A")

class B(A):
    def greet(self):
        super().greet()
        print("Hi from B")

B().greet()
```

21. Question: Can a class have multiple constructors?

Answer: Not directly, but you can use class methods as alternative constructors.

Code Example:

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

```
    @classmethod
    def from_birth_year(cls, name, year):
        return cls(name)

p = Person.from_birth_year("Alice", 1990)
```

22. Question: What is a dataclass?

Answer: A decorator that automatically generates init, repr, eq, etc.

Code Example:

```python
from dataclasses import dataclass

@dataclass
class Point:
    x: int
    y: int
```

```
p = Point(3, 4)
print(p)
```

23. Question: How do you create a singleton class?

Answer: By overriding new to return the same instance.

Code Example:

```python
class Singleton:
    _instance = None
    def __new__(cls):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
        return cls._instance
```

24. Question: What are slots?

Answer: They restrict attributes and save memory by using a fixed layout.

Code Example:

```python
class Point:
    __slots__ = ['x', 'y']

    def __init__(self, x, y):
        self.x = x
        self.y = y

p = Point(1, 2)
```

25. Question: How do you check if an object is an instance of a class?

Answer: Using isinstance().

Code Example:

```python
class Animal: pass
class Dog(Animal): pass

d = Dog()
```

print(isinstance(d, Dog))

print(isinstance(d, Animal))

```

26. Question: How do you check if a class is a subclass?

Answer: Using issubclass().

Code Example:

```python

class A: pass

class B(A): pass

print(issubclass(B, A))

```

27. Question: What is an abstract base class?

Answer: A class that cannot be instantiated and defines interface.

Code Example:

```python

from abc import ABC, abstractmethod

```
class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Square(Shape):
    def __init__(self, side):
        self.side = side
    def area(self):
        return self.side ** 2
```

28. Question: How do you make a class iterable?

Answer: Implement iter and next methods.

Code Example:

```python
class Range:
    def __init__(self, stop):
        self.stop = stop
```

```
    def __iter__(self):
        self.n = 0
        return self

    def __next__(self):
        if self.n < self.stop:
            self.n += 1
            return self.n - 1
        raise StopIteration

for i in Range(3):
    print(i)
```

29. Question: How do you make an object callable?

Answer: Implement the call method.

Code Example:

```python
```

```
class Greeter:
    def __call__(self, name):
        print(f"Hello {name}")

g = Greeter()
g("Alice")
```

30. Question: What is a metaclass?

Answer: A class of a class that defines how a class behaves.

Code Example:

```python
class Meta(type):
    pass

class MyClass(metaclass=Meta):
    pass

print(type(MyClass))
```

```

31. Question: What is encapsulation?

Answer: Bundling data and methods and restricting access to some components.

Code Example:

```python
class BankAccount:
    def __init__(self):
        self.__balance = 0

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
```

32. Question: What is abstraction?

Answer: Hiding complex implementation details and showing only essential features.

Code Example:
```

```python
from abc import ABC, abstractmethod

class Vehicle(ABC):
    @abstractmethod
    def start(self):
        pass

class Car(Vehicle):
    def start(self):
        print("Car started")
```

33. Question: How do you create a read-only attribute?

Answer: Use property without a setter.

Code Example:

```python
class Circle:
    def __init__(self, radius):
```

```
        self._radius = radius

    @property
    def radius(self):
        return self._radius

c = Circle(5)
print(c.radius)
```

34. Question: What is name mangling?

Answer: Python changes __attribute to _ClassName__attribute to make it private.

Code Example:

```python
class Test:
    def __secret(self):
        print("Secret")

t = Test()
```

t._Test__secret()

```

35. Question: What is composition?

Answer: A class contains objects of other classes as attributes.

Code Example:

```python

class Engine:

    def start(self):

        print("Engine on")


class Car:

    def __init__(self):

        self.engine = Engine()


    def start(self):

        self.engine.start()


Car().start()

```

36. Question: What is duck typing?

Answer: If it walks like a duck and quacks like a duck, it's a duck.

Code Example:

```python
class Duck:
    def quack(self):
        print("Quack")

class Person:
    def quack(self):
        print("I'm quacking")

def make_quack(obj):
    obj.quack()

make_quack(Duck())
make_quack(Person())
```
```

```

37. Question: What is the MRO?

Answer: Method Resolution Order - the order Python looks for methods in inheritance.

Code Example:

```python

class A: pass

class B(A): pass

class C(A): pass

class D(B, C): pass

print(D.__mro__)
```

38. Question: What is str vs repr?

Answer: str for readable output, repr for unambiguous representation.

Code Example:

```python

class Point:

```
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __str__(self):
        return f"({self.x},{self.y})"
    def __repr__(self):
        return f"Point({self.x},{self.y})"

p = Point(1, 2)
print(str(p))
print(repr(p))
```

39. Question: How do you implement equality comparison?

Answer: Override eq method.

Code Example:

```python
class Person:
    def __init__(self, id):
```

```
        self.id = id
    def __eq__(self, other):
        return self.id == other.id

p1 = Person(1)
p2 = Person(1)
print(p1 == p2)
```

40. Question: How do you implement length for a custom class?

Answer: Implement len method.

Code Example:

```python
class Team:
    def __init__(self):
        self.members = ["A", "B", "C"]
    def __len__(self):
        return len(self.members)
```

```
t = Team()
print(len(t))
```

41. Question: How do you make an object support indexing?

Answer: Implement getitem and setitem.

Code Example:

```python
class MyList:
    def __init__(self):
        self.data = [10, 20, 30]
    def __getitem__(self, i):
        return self.data[i]

ml = MyList()
print(ml[1])
```

42. Question: What is a descriptor?

Answer: An object that implements get, set, or delete for attribute access.

Code Example:

```python
class Descriptor:
    def __get__(self, obj, objtype=None):
        return 42

class MyClass:
    attr = Descriptor()

m = MyClass()
print(m.attr)
```

43. Question: How do you create an immutable object?

Answer: Use frozen=True in dataclass or slots with no setters.

Code Example:

```python
```

```
from dataclasses import dataclass


@dataclass(frozen=True)
class Point:
    x: int
    y: int


p = Point(1, 2)
```

44. Question: What is a class decorator?

Answer: A function that takes a class and returns a new class.

Code Example:

```python
def add_method(cls):
    cls.new_method = lambda self: print("Added")
    return cls


@add_method
```

```
class MyClass:
    pass

MyClass().new_method()
```

45. Question: How do you use Enum with classes?

Answer: Inherit from Enum to create named constants.

Code Example:

```python
from enum import Enum

class Color(Enum):
    RED = 1
    GREEN = 2

print(Color.RED)
```

46. Question: What is the difference between classmethod and staticmethod?

Answer: classmethod gets the class, staticmethod gets nothing.

Code Example:

```python
class Example:
    @classmethod
    def cm(cls):
        print("Class:", cls)
    @staticmethod
    def sm():
        print("No args")
```

47. Question: Can you inherit from built-in types?

Answer: Yes, to extend their functionality.

Code Example:

```python
class MyDict(dict):
```

```
    def hello(self):
        print("Hello")

d = MyDict()
d.hello()
```

48. Question: How do you implement a context manager in a class?

Answer: Implement enter and exit.

Code Example:

```python
class Managed:
    def __enter__(self):
        print("Entered")
        return self
    def __exit__(self, exc_type, exc, tb):
        print("Exited")

with Managed():
```

```
    print("Inside")
```

49. Question: What is a protocol in Python?

Answer: A set of methods an object must support (structural subtyping).

Code Example:

```python
from typing import Protocol

class Drawable(Protocol):
    def draw(self): ...

class Circle:
    def draw(self):
        print("Circle")

def render(obj: Drawable):
    obj.draw()
```

render(Circle())
```

50. Question: How do you create a factory method?

Answer: Use a class method that returns an instance.

Code Example:

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

    @classmethod
    def create_anonymous(cls):
        return cls("Anonymous")

p = Person.create_anonymous()
print(p.name)
```
```

51. Question: How do you delete an attribute?

Answer: Use del or implement delattr.

Code Example:

```python
class Test:
    def __init__(self):
        self.x = 10

t = Test()
del t.x
```

52. Question: What is init_subclass?

Answer: A class method called when a subclass is created.

Code Example:

```python
class Base:
    def __init_subclass__(cls, **kwargs):
        print(f"Subclass created: {cls}")
```

```
class Sub(Base):
    pass
```

53. Question: What is a mixin?

Answer: A class designed to be inherited to add functionality.

Code Example:

```python
class JsonMixin:
    def to_json(self):
        return {"data": "json"}

class User(JsonMixin):
    pass

u = User()
print(u.to_json())
```

54. Question: How do you implement hash for a custom class?

Answer: Implement hash and eq.

Code Example:

```python
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __eq__(self, other):
        return self.x == other.x and self.y == other.y
    def __hash__(self):
        return hash((self.x, self.y))
```

55. Question: What is the diamond problem?

Answer: When a class inherits from two classes that both inherit from a common base. Python uses MRO to resolve it.

Code Example:

```python
class A:
    def show(self):
        print("A")

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

D().show()
```

56. Question: How do you prevent attribute creation?

Answer: Use slots or override setattr.

Code Example:

```python
class Fixed:
```

```
    __slots__ = ['name']
    def __init__(self):
        self.name = "test"
```

57. Question: What is a property deleter?

Answer: A method that runs when an attribute is deleted.

Code Example:

```python
class Test:
    @property
    def x(self):
        return 42
    @x.deleter
    def x(self):
        print("Deleted")

t = Test()
del t.x
```

```

58. Question: How do you make a class hashable?

Answer: Implement eq and hash.

Code Example:

```python
class Person:
    def __init__(self, name):
        self.name = name
    def __eq__(self, other):
        return self.name == other.name
    def __hash__(self):
        return hash(self.name)
```

59. Question: What is the difference between new and init?

Answer: new creates the object, init initializes it. new runs first.

Code Example:
```

```python
class Test:
    def __new__(cls):
        print("New")
        return super().__new__(cls)
    def __init__(self):
        print("Init")

Test()
```

60. Question: How do you implement a custom iterator?

Answer: Create a class with iter and next.

Code Example:

```python
class Counter:
    def __init__(self, max):
        self.max = max
    def __iter__(self):
```

```
        self.n = 0
        return self
    def __next__(self):
        self.n += 1
        if self.n > self.max:
            raise StopIteration
        return self.n - 1
```

61. Question: What is a class variable vs instance variable?

Answer: Class variable is shared, instance variable is per object.

Code Example:

```python
class Test:
    shared = 0
    def __init__(self):
        self.own = 1
```

```
t1 = Test()

t2 = Test()

Test.shared = 42

print(t1.shared, t2.shared)
```

62. Question: How do you access a class variable in a method?

Answer: Using self.__class__.var or ClassName.var.

Code Example:

```python
class Counter:
    count = 0
    def __init__(self):
        self.__class__.count += 1
```

63. Question: Can static methods access class attributes?

Answer: No, they don't receive cls.

Code Example:

```python
class Test:
    x = 10
    @staticmethod
    def get_x():
        print(Test.x)  # Works if using class name
```

64. Question: How do you make a class support addition and multiplication?

Answer: Implement add and mul.

Code Example:

```python
class Num:
    def __init__(self, v):
        self.v = v
    def __add__(self, other):
        return Num(self.v + other.v)
    def __mul__(self, other):
```

```
        return Num(self.v * other.v)
```

65. Question: What is del?

Answer: The destructor method called when object is garbage collected.

Code Example:

```python
class Test:
    def __del__(self):
        print("Object destroyed")

t = Test()
del t
```

66. Question: How do you implement comparison operators?

Answer: Implement lt, gt, le, ge, eq.

Code Example:

```python
class Person:
    def __init__(self, age):
        self.age = age
    def __lt__(self, other):
        return self.age < other.age
```

67. Question: What is a namedtuple?

Answer: A factory function that creates a tuple subclass with named fields.

Code Example:

```python
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])
p = Point(1, 2)
print(p.x)
```

68. Question: What is a frozen dataclass?

Answer: An immutable dataclass.

Code Example:

```python
@dataclass(frozen=True)
class Config:
    debug: bool = False
```

69. Question: How do you create a class with default values?

Answer: Set defaults in init or use dataclass.

Code Example:

```python
class User:
    def __init__(self, name="Guest"):
        self.name = name
```

70. Question: How do you document a class?

Answer: Use docstring at class level.

Code Example:

```python
class Calculator:
    """A simple calculator class"""
    def add(self, a, b):
        return a + b
```

71. Question: How do you create a private method?

Answer: Prefix with double underscore.

Code Example:

```python
class Test:
    def __private(self):
        print("Private")
    def public(self):
        self.__private()
```

72. Question: What is the Liskov Substitution Principle?

Answer: Subtypes must be substitutable for their base types.

Code Example:

```python
# Good example: all shapes can calculate area
class Rectangle:
    def area(self): return self.w * self.h
```

73. Question: What is the Open-Closed Principle?

Answer: Classes should be open for extension but closed for modification.

Code Example:

```python
# Extend by inheritance instead of modifying base class
class Shape: pass
class Circle(Shape): pass
```

74. Question: What is the Single Responsibility Principle?

Answer: A class should have only one reason to change.

Code Example:

```python

class User:  # handles user data

class UserSaver:  # handles saving

```

https://www.linkedin.com/in/mohd-maaz-08691a366/

75. Question: What is dependency injection?

Answer: Passing dependencies instead of creating them inside.

Code Example:

```python

class Database:

    def save(self):

        print("Saved")


class UserService:
```

```
    def __init__(self, db):
        self.db = db
```

76. Question: How do you use class as a decorator?

Answer: Implement call in the class.

Code Example:

```python
class Decorator:
    def __init__(self, func):
        self.func = func
    def __call__(self, *args):
        print("Before")
        self.func(*args)
        print("After")

@Decorator
def hello():
    print("Hello")
```

hello()
```

77. Question: What is a classmethod used as constructor?

Answer: Common pattern for alternative constructors.

Code Example:

```python
class Date:
    def __init__(self, y, m, d):
        self.y = y
        self.m = m
        self.d = d

    @classmethod
    def from_string(cls, s):
        y, m, d = map(int, s.split('-'))
        return cls(y, m, d)
```
```

78. Question: How do you copy an object?

Answer: Use copy.copy() or copy.deepcopy() or implement copy.

Code Example:

```python
import copy
original = SomeClass()
copy = copy.deepcopy(original)
```

79. Question: What is a class factory?

Answer: A function that returns a class.

Code Example:

```python
def create_class(name):
    return type(name, (), {})
MyClass = create_class("MyClass")
```

80. Question: How do you make a class support boolean evaluation?

Answer: Implement bool or nonzero.

Code Example:

```python
class AlwaysTrue:
    def __bool__(self):
        return True
print(bool(AlwaysTrue()))
```

81. Question: How do you make a class support in operator?

Answer: Implement contains.

Code Example:

```python
class Bag:
    def __init__(self):
        self.items = [1, 2, 3]
    def __contains__(self, item):
```

```
        return item in self.items

print(2 in Bag())
```

82. Question: What is a class with only class methods called?

Answer: Utility or helper class, often stateless.

Code Example:

```python
class StringUtils:
    @classmethod
    def reverse(cls, s):
        return s[::-1]
```

83. Question: How do you prevent inheritance?

Answer: Not directly possible, but you can raise error in init_subclass.

Code Example:

```python
```

```
class Final:
    def __init_subclass__(cls, **kwargs):
        raise TypeError("Cannot subclass Final")
```

84. Question: What is a data hiding?

Answer: Hiding internal attributes using private variables.

Code Example:

```python
class Safe:
    def __init__(self):
        self.__secret = "hidden"
    def reveal(self):
        return self.__secret
```

85. Question: How do you create a cached property?

Answer: Use functools.cached_property.

Code Example:

```python
from functools import cached_property


class Circle:
    def __init__(self, r):
        self.r = r
    @cached_property
    def area(self):
        return 3.14 * self.r ** 2
```

86. Question: What is a classmethod vs instancemethod?

Answer: classmethod receives the class, instancemethod receives the instance.

Code Example:

```python
class Demo:
    @classmethod
    def class_method(cls):
```

```
        print(cls)
    def instance_method(self):
        print(self)
```

87. Question: How do you implement a singleton with decorator?

Answer: Create a decorator that caches the instance.

Code Example:

```python
def singleton(cls):
    instances = {}
    def get_instance(*args, **kwargs):
        if cls not in instances:
            instances[cls] = cls(*args, **kwargs)
        return instances[cls]
    return get_instance

@singleton
class DB: pass
```

```

88. Question: What is a class with slots vs without?

Answer: slots saves memory and prevents new attributes.

Code Example:

```python

class Normal: pass

class Optimized:

    __slots__ = ['x']
```

89. Question: How do you make a class support await?

Answer: Implement await.

Code Example:

```python

class AsyncTask:

    def __await__(self):

        return (yield from some_coro())
```
```

90. Question: What is the type of a class?

Answer: Its metaclass, usually type.

Code Example:

```python
class MyClass:
    pass
print(type(MyClass))
```

91. Question: How do you dynamically add a method to a class?

Answer: Assign a function to the class.

Code Example:

```python
class A: pass
def hello(self):
    print("Hello")
A.hello = hello
A().hello()
```

```

92. Question: What is a classmethod factory?

Answer: A class method that creates and returns instances.

Code Example:

```python
class Color:
    @classmethod
    def red(cls):
        return cls(255, 0, 0)
```

93. Question: How do you implement a custom repr that eval can use?

Answer: Return a string that recreates the object.

Code Example:

```python
class Point:
    def __init__(self, x, y):
```
```

```
        self.x = x
        self.y = y
    def __repr__(self):
        return f"Point({self.x}, {self.y})"
```

94. Question: What is the difference between new and init?

Answer: new allocates memory, init initializes the object.

Code Example:

```python
class Test:
    def __new__(cls):
        obj = super().__new__(cls)
        print("Created")
        return obj
    def __init__(self):
        print("Initialized")
```

95. Question: How do you make a class support sorting?

Answer: Implement lt or use key with functools.total_ordering.

Code Example:

```python
from functools import total_ordering

@total_ordering
class Number:
    def __init__(self, n):
        self.n = n
    def __lt__(self, other):
        return self.n < other.n
```

96. Question: What is a class attribute vs instance attribute access?

Answer: Instance attributes shadow class attributes if same name.

Code Example:

```python
class Test:
    x = 1
t = Test()
t.x = 100
print(t.x)      # 100
print(Test.x)   # 1
```

97. Question: How do you create a class with private constructor?

Answer: Raise exception in init or use factory.

Code Example:

```python
class Private:
    def __init__(self):
        raise RuntimeError("Use create()")
    @classmethod
    def create(cls):
```

```
        return super().__new__(cls)
```

98. Question: What is a class with both class and instance methods?

Answer: Common pattern for utility and instance operations.

Code Example:

```python
class StringUtil:
    @classmethod
    def clean(cls, s):
        return s.strip()
    def process(self, s):
        return s.upper()
```

99. Question: How do you implement a class that supports len and getitem?

Answer: Implement len and getitem.

Code Example:

```python
class MySeq:
    def __init__(self, data):
        self.data = data
    def __len__(self):
        return len(self.data)
    def __getitem__(self, i):
        return self.data[i]
```

100. Question: What is the base class of all classes?

Answer: object is the ultimate base class in Python.

Code Example:

```python
class MyClass:
    pass
print(MyClass.__bases__)
print(MyClass.__bases__[0].__bases__)
```

```

101. Question: How do you create a class with custom attribute access?

Answer: Implement getattribute, getattr, setattr.

Code Example:

```python

class Logging:

    def __getattribute__(self, name):

        print(f"Accessing {name}")

        return super().__getattribute__(name)

```

102. Question: What is a class decorator that adds attributes?

Answer: A function that modifies the class dictionary.

Code Example:

```python

def add_attr(cls):

    cls.extra = "hello"

```
    return cls

@add_attr
class Test: pass
print(Test.extra)
```

103. Question: How do you implement a class that prints on creation?

Answer: Use init or new.

Code Example:

```python
class Loud:
    def __new__(cls):
        print("Creating instance")
        return super().__new__(cls)
```

104. Question: What is a class that cannot be instantiated?

Answer: Abstract base class with abstract methods.

Code Example:

```python
from abc import ABC


class Abstract(ABC):
    pass
```

105. Question: How do you create a class with default mutable arguments safely?

Answer: Use None and initialize inside.

Code Example:

```python
class Safe:
    def __init__(self, items=None):
        self.items = items or []
```

106. Question: What is a class with a custom metaclass that logs creation?

Answer: Metaclass with new or init_subclass.

Code Example:

```python
class LoggingMeta(type):
    def __call__(cls, *args, **kwargs):
        print(f"Creating {cls.__name__}")
        return super().__call__(*args, **kwargs)

class User(metaclass=LoggingMeta):
    pass
```

107. Question: How do you make a class support context management?

Answer: Implement enter and exit.

Code Example:

```python
class TempFile:
```

```
    def __enter__(self):
        print("Opened")
        return self
    def __exit__(self, *args):
        print("Closed")
```

108. Question: What is a class that acts as a function?

Answer: Implement call.

Code Example:

```python
class Multiplier:
    def __call__(self, x):
        return x * 2

double = Multiplier()
print(double(5))
```

109. Question: How do you create a class with ordered attributes?

Answer: Use dataclass with order=True.

Code Example:

```python
@dataclass(order=True)
class Person:
    age: int
    name: str
```

110. Question: What is a class with a custom attribute validation?

Answer: Use property setters or descriptors.

Code Example:

```python
class Validated:
    def __init__(self):
        self._value = 0
    @property
```

```
    def value(self):
        return self._value
    @value.setter
    def value(self, v):
        if v < 0:
            raise ValueError
        self._value = v
```

111. Question: How do you create a class that logs attribute access?

Answer: Override getattribute.

Code Example:

```python
class Logger:
    def __getattribute__(self, name):
        print(f"Getting {name}")
        return super().__getattribute__(name)
```

112. Question: What is a class with a custom hash that never collides?

Answer: Not possible to guarantee, but can minimize collisions.

Code Example:

```python
class Unique:
    def __init__(self, id):
        self.id = id
    def __hash__(self):
        return self.id
```

113. Question: How do you create a class that counts instances?

Answer: Use class variable incremented in init.

Code Example:

```python
class Counter:
    count = 0
```

```
    def __init__(self):
        Counter.count += 1
```

114. Question: What is a class that returns a different type from a method?

Answer: Possible, but breaks polymorphism if not careful.

Code Example:

```python
class Strange:
    def get_number(self):
        return "string"
```

115. Question: How do you make a class support pickling?

Answer: Usually works automatically, or implement getstate/setstate.

Code Example:

```python
```

```
import pickle
class Picklable:
    def __init__(self, data):
        self.data = data
pickle.dumps(Picklable("data"))
```

116. Question: What is a class with a custom metaclass that adds methods?

Answer: Metaclass can modify class dict.

Code Example:

```python
class AutoMethod(type):
    def __new__(cls, name, bases, dct):
        dct['hello'] = lambda self: print("Hi")
        return super().__new__(cls, name, bases, dct)
```

117. Question: How do you create a class that prevents subclassing?

Answer: Use metaclass or init_subclass raising error.

Code Example:

```python
class Sealed:
    def __init_subclass__(cls, **kwargs):
        raise TypeError("Cannot inherit")
```

118. Question: What is a class that supports both list and dict behavior?

Answer: Inherit from UserList or UserDict or implement manually.

Code Example:

```python
from collections import UserDict

class MyDict(UserDict):
    def hello(self):
        print("Hello")
```

119. Question: How do you create a class with a custom attribute name validation?

Answer: Override setattr.

Code Example:

```python
class Strict:
    def __setattr__(self, name, value):
        if not name.isalpha():
            raise ValueError
        super().__setattr__(name, value)
```

120. Question: What is a class that automatically converts to JSON?

Answer: Implement to_dict or to_json method.

Code Example:

```python
import json
```

```
class Jsonable:
    def __init__(self, name):
        self.name = name
    def to_json(self):
        return json.dumps({"name": self.name})
```

121. Question: How do you create a class with a custom string representation for debugging?

Answer: Override repr.

Code Example:

```python
class Debug:
    def __init__(self, x):
        self.x = x
    def __repr__(self):
        return f"Debug(x={self.x})"
```

122. Question: What is a class that supports dynamic attribute names?

Answer: All classes do by default.

Code Example:

```python
class Dynamic:
    pass


obj = Dynamic()
obj.any_name = "value"
```

123. Question: How do you create a class that logs every method call?

Answer: Use a metaclass or decorator.

Code Example:

```python
def log_calls(cls):
    for name, method in cls.__dict__.items():
        if callable(method) and not name.startswith("_"):
```

```
            # decorate method
            pass
    return cls
```

124. Question: What is a class that behaves like a function and has state?

Answer: Callable object with call method.

Code Example:

```python
class CounterFunc:
    def __init__(self):
        self.count = 0
    def __call__(self):
        self.count += 1
        return self.count

c = CounterFunc()
print(c(), c())
```

125. Question: How do you create a class with a custom attribute deletion behavior?

Answer: Implement delattr.

Code Example:

```python
class Logged:
    def __delattr__(self, name):
        print(f"Deleting {name}")
        super().__delattr__(name)
```

126. Question: What is a class that supports both addition and subtraction?

Answer: Implement add and sub.

Code Example:

```python
class Vector:
    def __init__(self, x):
        self.x = x
```

```
    def __add__(self, other):
        return Vector(self.x + other.x)
    def __sub__(self, other):
        return Vector(self.x - other.x)
```

127. Question: How do you create a class that supports matrix multiplication?

Answer: Implement matmul (@ operator).

Code Example:

```python
class Mat:
    def __matmul__(self, other):
        return "multiplied"
```

128. Question: What is a class that automatically tracks modified attributes?

Answer: Override setattr and keep a set.

Code Example:

```python
class Tracked:
    def __init__(self):
        self._modified = set()
    def __setattr__(self, name, value):
        if name != "_modified":
            self._modified.add(name)
        super().__setattr__(name, value)
```

129. Question: How do you create a class with a custom pickling protocol?

Answer: Implement getstate and setstate.

Code Example:

```python
class CustomPickle:
    def __getstate__(self):
        return {"data": "saved"}
    def __setstate__(self, state):
        print("Restored")
```

```

130. Question: What is a class that supports both int and str conversion?

Answer: Implement int and str.

Code Example:

```python
class Dual:
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return str(self.value)
    def __int__(self):
        return int(self.value)
```

131. Question: How do you create a class that validates on assignment?

Answer: Use property setters or descriptors.

Code Example:
```

```python
class Validated:
    @property
    def age(self):
        return self._age
    @age.setter
    def age(self, value):
        if not 0 <= value <= 150:
            raise ValueError("Invalid age")
        self._age = value
```

132. Question: What is a class that supports the with statement and is reusable?

Answer: Implement enter and exit correctly.

Code Example:

```python
class Reusable:
    def __enter__(self):
        return self
```

```
    def __exit__(self, exc_type, exc, tb):
        pass  # cleanup if needed
```

133. Question: How do you create a class that prints creation time?

Answer: Store time in init.

Code Example:

```python
from datetime import datetime

class Timed:
    def __init__(self):
        self.created = datetime.now()
```

134. Question: What is a class that supports both equality and inequality?

Answer: Implement eq and ne.

Code Example:

```python
class Person:
    def __init__(self, name):
        self.name = name
    def __eq__(self, other):
        return self.name == other.name
    def __ne__(self, other):
        return not self.__eq__(other)
```

135. Question: How do you create a class that supports truthiness based on attribute?

Answer: Implement bool.

Code Example:

```python
class NonEmpty:
    def __init__(self, items):
        self.items = items
    def __bool__(self):
        return len(self.items) > 0
```

```

136. Question: What is a class that supports the format() function?

Answer: Implement format.

Code Example:

```python
class Money:
    def __init__(self, amount):
        self.amount = amount
    def __format__(self, spec):
        return f"${self.amount:.2f}"
```

137. Question: How do you create a class that supports reversed()?

Answer: Implement reversed.

Code Example:

```python
class MyRange:

```
    def __init__(self, n):
        self.n = n
    def __reversed__(self):
        return range(self.n - 1, -1, -1)
```

138. Question: What is a class that supports the in keyword on custom condition?

Answer: Implement contains.

Code Example:

```python
class SpecialSet:
    def __init__(self):
        self.data = {1, 2, 3}
    def __contains__(self, item):
        return item > 0
```

139. Question: How do you create a class that supports slicing?

Answer: Implement getitem with slice support.

Code Example:

```python
class Sliceable:
    def __getitem__(self, key):
        if isinstance(key, slice):
            return list(range(10))[key]
        return "item"
```

140. Question: What is a class that supports the abs() function?

Answer: Implement abs.

Code Example:

```python
class Number:
    def __init__(self, x):
        self.x = x
    def __abs__(self):
        return abs(self.x)
```

```

141. Question: How do you create a class that supports round(), floor(), ceil()?

Answer: Implement round, floor, ceil via magic methods.

Code Example:

```python
class Rounded:
    def __init__(self, x):
        self.x = x
    def __round__(self, n=None):
        return round(self.x, n)
```

142. Question: What is a class that supports the divmod() function?

Answer: Implement divmod.

Code Example:

```python
class Divisible:

```
    def __init__(self, x):
        self.x = x
    def __divmod__(self, other):
        return divmod(self.x, other)
```

143. Question: How do you create a class that supports the pow() function?

Answer: Implement pow.

Code Example:

```python
class Power:
    def __init__(self, base):
        self.base = base
    def __pow__(self, exponent):
        return self.base ** exponent
```

144. Question: What is a class that supports the bytes() function?

Answer: Implement bytes.

Code Example:

```python
class ToBytes:
    def __bytes__(self):
        return b"hello"
print(bytes(ToBytes()))
```

145. Question: How do you create a class that supports the hash() function safely?

Answer: Implement both eq and hash consistently.

Code Example:

```python
class Hashable:
    def __init__(self, value):
        self.value = value
    def __hash__(self):
        return hash(self.value)
    def __eq__(self, other):
```

```
        return self.value == other.value
```

146. Question: What is a class that supports the dir() function customization?

Answer: Implement dir.

Code Example:

```python
class CustomDir:
    def __dir__(self):
        return ['custom', 'list']
print(dir(CustomDir()))
```

147. Question: How do you create a class that supports the complex() function?

Answer: Implement complex.

Code Example:

```python
class ComplexNum:
```

```
    def __complex__(self):
        return 3 + 4j
```

148. Question: What is a class that supports the oct(), hex(), bin() functions?

Answer: Implement oct, hex, bin.

Code Example:

```python
class Num:
    def __init__(self, n):
        self.n = n
    def __index__(self):
        return self.n
```

149. Question: How do you create a class that supports the format spec with custom codes?

Answer: Implement format.

Code Example:

```python
class Temperature:
    def __init__(self, c):
        self.c = c
    def __format__(self, format_spec):
        if format_spec == 'f':
            return f"{self.c * 9/5 + 32}F"
        return f"{self.c}C"
```

150. Question: What is the ultimate base class in Python?

Answer: object. Every class inherits from it directly or indirectly.

Code Example:

```python
class Empty:
    pass

print(Empty.__mro__)
```

```
print(object in Empty.__mro__)
```