Certainly! Here are 10 medium to tough multiple choice questions on Object-Oriented Programming (OOP) in Python:

1. What is the output of the following code?

python

Copy

class A:

def \_\_init\_\_(self):

self.x = 1

class B(A):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.y = 2

b = B()

print(b.x, b.y)

a) 1 2 b) Error c) None 2 d) 1 None

1. Which of the following is true about multiple inheritance in Python? a) Python does not support multiple inheritance b) Python uses C3 linearization algorithm for method resolution c) The order of base classes doesn't matter in multiple inheritance d) Multiple inheritance always leads to the diamond problem
2. What is the purpose of the \_\_slots\_\_ attribute in a Python class? a) To define class methods b) To restrict the creation of new attributes c) To define class variables d) To create immutable objects
3. What will be the output of the following code?

python

Copy

class MyClass:

def \_\_init\_\_(self):

self.\_x = 5

@property

def x(self):

return self.\_x

@x.setter

def x(self, value):

if value < 0:

self.\_x = 0

elif value > 10:

self.\_x = 10

else:

self.\_x = value

obj = MyClass()

obj.x = 15

print(obj.x)

a) 15 b) 10 c) 5 d) AttributeError

1. Which of the following is true about abstract base classes in Python? a) They are created using the abstract keyword b) They can be instantiated directly c) They are created using the abc module d) They cannot contain any method implementations
2. What is the output of the following code?

python

Copy

class A:

def \_\_init\_\_(self):

print("A", end="")

class B(A):

def \_\_init\_\_(self):

print("B", end="")

super().\_\_init\_\_()

class C(A):

def \_\_init\_\_(self):

print("C", end="")

super().\_\_init\_\_()

class D(B, C):

def \_\_init\_\_(self):

print("D", end="")

super().\_\_init\_\_()

D()

a) DBCA b) DBAC c) DCBA d) DACB

1. What is the purpose of the \_\_new\_\_ method in Python? a) To initialize a new instance b) To create and return a new instance c) To define class attributes d) To define instance methods
2. Which of the following is true about metaclasses in Python? a) They are regular classes that inherit from type b) They are used to create class objects c) They can modify class creation behavior d) All of the above
3. What will be the output of the following code?

python

Copy

class MyClass:

def \_\_init\_\_(self):

self.x = 5

def \_\_call\_\_(self):

return self.x \* 2

obj = MyClass()

print(obj())

a) 5 b) 10 c) TypeError d) None

1. What is the purpose of the @classmethod decorator in Python? a) To define a method that can be called on the class and its instances b) To define a static method c) To define a method that can access and modify class state d) To define a method that cannot be overridden in subclasses

Object-Oriented Programming:

1. Answer: a) 1 2 Explanation: The super().**init**() call in B's **init** method calls A's **init**, setting x to 1. Then y is set to 2.
2. Answer: b) Python uses C3 linearization algorithm for method resolution Explanation: Python uses the C3 linearization algorithm to determine the method resolution order in multiple inheritance.
3. Answer: b) To restrict the creation of new attributes Explanation: **slots** is used to explicitly declare data members and can help reduce memory usage.
4. Answer: b) 10 Explanation: The setter method limits the value of x to be between 0 and 10, so when 15 is assigned, it's set to 10.
5. Answer: c) They are created using the abc module Explanation: The abc (Abstract Base Classes) module is used to define abstract base classes in Python.
6. Answer: a) DBCA Explanation: The method resolution order for D is D -> B -> C -> A, and each **init** method prints its corresponding letter.
7. Answer: b) To create and return a new instance Explanation: **new** is responsible for creating and returning a new instance of the class.
8. Answer: d) All of the above Explanation: Metaclasses are classes for creating class objects, inherit from type, and can modify class creation behavior.
9. Answer: b) 10 Explanation: The **call** method makes the object callable. When called, it returns self.x \* 2, which is 5 \* 2 = 10.
10. Answer: c) To define a method that can access and modify class state Explanation: @classmethod decorator is used to define methods that can access and modify the class state.