**Module 12. Inheritance**

**TRUE/FALSE**

1. A subclass can override methods defined in its superclass.

Answer: True

1. Python supports method overloading in the same way as Java or C++.

Answer: False

1. In Python, a subclass automatically inherits all attributes and methods of its superclass.

Answer: True

1. The super() function is used to call the constructor of the subclass.

Answer: False

1. Inheritance helps avoid code duplication.

Answer: True

1. A subclass can inherit from multiple super classes in Python.

Answer: True

1. Method overriding occurs when a superclass provides a specific implementation for a method that is already defined in its subclass.

Answer: False

1. In UML diagrams, inheritance is shown with an arrow pointing from the superclass to the subclass.

Answer: False

1. The \_\_init\_\_ method of a superclass must be explicitly called in the \_\_init\_\_ method of a subclass.

Answer: True

1. The isinstance() function can be used to check if an object is an instance of a specific class or a subclass thereof.

Answer: True

1. A subclass cannot have additional attributes or methods beyond those inherited from the superclass.

Answer: False

1. Inheritance promotes code reusability.

Answer: True

1. Python classes cannot inherit from built-in types like list or dict.

Answer: False

1. Overriding a method in a subclass changes the behavior of that method for instances of the subclass only.

Answer: True

1. The super() function can only be used inside the \_\_init\_\_ method.

Answer: False

1. When a subclass method has the same name as a superclass method, the subclass method hides the superclass method.

Answer: True

1. Abstract classes in Python cannot be instantiated directly.

Answer: True

1. In Python, the @abstractmethod decorator is used to define methods that must be implemented by subclasses.

Answer: True

1. A subclass can call a method from its superclass using the super() function.

Answer: True

1. Inheritance allows a new class to inherit members of the class it extends.

Answer: True

**MULTIPLE CHOICE**

1. Which keyword is used to define a subclass in Python?

a. extends

b. inherits

c. subclass

d. class

Answer: d. class

1. In the following line of code, what is the name of the superclass?

*class Dog(Animal):*

a. Dog

b. Animal

c. class

d. None of these

Answer: b. Animal

1. Which of the following is NOT a benefit of inheritance?

a. Avoids code duplication

b. Makes code more intuitive

c. Increases code complexity

d. Promotes code reusability

Answer: c. Increases code complexity

1. In the following code, which method is overridden?

*class Car:*

*def print\_info(self):*

*print("Car information")*

*class Truck(Car):*

*def print\_info(self):*

*print("Truck information")*

a. \_\_init\_\_

b. print\_info

c. Car

d. Truck

Answer: b. print\_info

1. The super() function is used to:

a. Create a new class

b. Call methods from a superclass

c. Define a superclass

d. Override a method

Answer: b. Call methods from a superclass

1. Which of the following is the correct syntax for defining a class Car that inherits from Vehicle?

a. class Car[Vehicle]:

b. class Car: Vehicle

c. class Car(Vehicle):

d. class Vehicle(Car):

Answer: c. class Car(Vehicle):

1. What is the output of the following code?

*class Animal:*

*def eat(self):*

*print("Animal eats")*

*class Dog(Animal):*

*def bark(self):*

*print("Dog barks")*

*d = Dog()*

*d.eat()*

a. Dog barks

b. Animal eats

c. Error

d. None of these

Answer: b. Animal eats

1. What does the super() function do in a subclass?

a. Calls the subclass constructor

b. Calls the superclass constructor

c. Overrides a superclass method

d. Inherits attributes

Answer: b. Calls the superclass constructor

1. In a UML diagram, inheritance is typically represented by:

a. A solid line

b. A dashed line

c. A solid line with a filled arrowhead

d. A solid line with an open arrowhead

Answer: d. A solid line with an open arrowhead

1. What is the purpose of method overriding?

a. To create multiple methods with the same name in different classes

b. To replace a method in the superclass with a new implementation in the subclass

c. To inherit methods from another class

d. To hide methods in the superclass

Answer: b. To replace a method in the superclass with a new implementation in the subclass

1. Which of the following best describes the "is a" relationship in inheritance?

a. An object is a subclass of another object

b. An object is an instance of a class

c. An object has attributes

d. An object can call methods from another object

Answer: a. An object is a subclass of another object

1. How can you call the \_\_init\_\_ method of a superclass in a subclass?

a. self.\_\_init\_\_()

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

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

d. self.super().\_\_init\_\_()

Answer: b. super().\_\_init\_\_()

1. In the context of inheritance, what is a "subclass"?

a. A class that inherits from another class

b. A class that provides methods for another class

c. A class that is independent of other classes

d. A class that has no attributes or methods

Answer: a. A class that inherits from another class

1. What is method overloading?

a. A subclass method that replaces a superclass method

b. A superclass method that replaces a subclass method

c. Multiple methods with the same name but different parameters

d. A method that calls another method

Answer: c. Multiple methods with the same name but different parameters

1. In Python, how do you create a subclass that inherits from a superclass?

a. class Subclass(Superclass):

b. class Subclass extends Superclass:

c. class Subclass inherits Superclass:

d. class Subclass[Superclass]:

Answer: a. class Subclass(Superclass):

1. Which of the following statements is true about abstract methods in Python?

a. They must be defined in every class

b. They cannot be overridden

c. They must be implemented by subclasses

d. They can only be used in final classes

Answer: c. They must be implemented by subclasses

1. How can you check if an object is an instance of a specific class or subclass?

a. issubclass(object, class)

b. isinstance(object, class)

c. type(object) == class

d. object is class

Answer: b. isinstance(object, class)

1. What is the main purpose of inheritance in object-oriented programming?

a. To increase code complexity

b. To avoid code duplication and promote reusability

c. To create private attributes

d. To write longer code

Answer: b. To avoid code duplication and promote reusability

1. In Python, what is the base class for all classes?

a. Object

b. object

c. BaseClass

d. base

Answer: b. object

1. Which of the following is an example of method overriding?

a. Defining a method with the same name but different parameters

b. Defining a method with the same name and same parameters in a subclass

c. Defining a new method in a subclass

d. Using the super() function to call a superclass method

Answer: b. Defining a method with the same name and same parameters in a subclass

1. What is the output of the following code?

*class Parent:*

*def show(self):*

*print("Parent method")*

*class Child(Parent):*

*def show(self):*

*print("Child method")*

*c = Child()*

*c.show()*

a. Parent method

b. Child method

c. Error

d. None of these

Answer: b. Child method

1. Which of the following statements about the super() function is correct?

a. It can only be used in the \_\_init\_\_ method

b. It is used to call methods from a subclass

c. It is used to call methods from a superclass

d. It can only be used in abstract classes

Answer: c. It is used to call methods from a superclass

1. In the context of inheritance, what does the term "polymorphism" mean?

a. The ability of different objects to respond to the same method in different ways

b. The creation of multiple subclasses from a single superclass

c. The ability of a subclass to override a superclass method

d. The use of the super() function

Answer: a. The ability of different objects to respond to the same method in different ways

1. What is the main advantage of using inheritance in Python?

a. It increases code length

b. It promotes code reuse and organization

c. It makes code harder to understand

d. It prevents the use of methods from other classes

Answer: b. It promotes code reuse and organization