MCQs on OOP, Exceptions & Modules in Python

Prepared by: Abdelrahman Eldaba

بسم الله الرحمن الرحيم

الحمد لله رب العالمين، والصلاة والسلام على سيدنا محمد وعلى آله وصحبه أجمعين

لقد قمت بإعداد هذه الأسئلة لعلها تكون طريقة لفهم ما تعلمناه في لغة البرمجة

PYTHON

متمنياً أن ينال هذا العمل إعجابكم وأسأل الله تعالى أن يوفقنا في هذه الرحلة التعليمية

وأن يجعلها في ميزان حسناتنا

والله ولي التوفيق

IN THE NAME OF ALLAH, THE MOST GRACIOUS, THE MOST MERCIFUL

ALL PRAISE IS DUE TO ALLAH, THE LORD OF ALL WORLDS, AND MAY PEACE AND BLESSINGS BE UPON OUR PROPHET MUHAMMAD AND HIS FAMILY AND COMPANIONS

I HAVE PREPARED THESE QUESTIONS IN THE PROGRAMMING LANGUAGE PYTHON, HOPING THAT THEY WILL SERVE AS A WAY TO UNDERSTAND WHAT WE HAVE LEARNED. I HOPE YOU FIND THIS WORK ENJOYABLE, AND I ASK ALLAH ALMIGHTY TO GRANT US SUCCESS IN THIS EDUCATIONAL JOURNEY AND TO MAKE IT A MEANS OF ACCUMULATING GOOD DEEDS.

AND ALLAH IS THE SOURCE OF ALL SUCCESS

1. What does OOP stand for?
    a) Object Oriented Programming
    b) Object Order Protocol
    c) Object Oriented Process
    d) None of the above

2. Which of the following is not a principle of OOP?
    a) Inheritance
    b) Polymorphism
    c) Abstraction
    d) Imperative programming

3. What is the purpose of encapsulation in OOP?
    a) To hide the implementation details of a class
    b) To allow multiple inheritance
    c) To create an interface for the class
    d) None of the above

4. What is the syntax for defining a class in Python?
    a) class ClassName:
    b) ClassName:
    c) define class ClassName:
    d) class ClassName():

5. Which keyword is used to define a method within a class?
    a) def
    b) function
    c) method
    d) subroutine

6. Which of the following is not an access modifier in Python?
    a) private
    b) protected
    c) public
    d) final

7. What is the purpose of inheritance in OOP?
    a) To create new classes from existing ones
    b) To allow multiple instances of a class
    c) To hide implementation details of a class
    d) None of the above

8. Which keyword is used to indicate that a method is a class method in Python?
    a) static
    b) classmethod
    c) method
    d) decorator

9. What is the purpose of the init method in Python?
    a) To create a new instance of a class
    b) To initialize the attributes of a class
    c) To define a class method
    d) None of the above

10. Which of the following is a valid way to access a class attribute in Python?
    a) ClassName.attribute
    b) ClassName->attribute
    c) ClassName[attribute]
    d) None of the above

11. What is the purpose of the self parameter in Python class methods?
    a) It refers to the class object itself
    b) It refers to the first argument passed to the method
    c) It refers to the instance of the class on which the method was called
    d) None of the above

12. What is the syntax for creating an instance of a class in Python?
    a) ClassName(object)
    b) ClassName()
    c) object.ClassName()
    d) None of the above

13. What is the purpose of the super() function in Python?

a) It is used to call a method in a superclass
b) It is used to create a new instance of a class
c) It is used to define a static method
d) None of the above

14. What is the purpose of the @staticmethod decorator in Python?

a) It indicates that a method is a class method
b) It indicates that a method is a static method
c) It indicates that a method is a generator function
d) None of the above

15. Which of the following is an example of polymorphism in Python?

a) A class inheriting from multiple parent classes
b) A method having different implementations in different subclasses
c) A class having multiple instances
d) None of the above

16. What is the purpose of the isinstance() function in Python?

a) To check if an object is an instance of a class
b) To check if a method is a class method
c) To check if a method is a static method
d) None of the above

17. Which of the following is an example of encapsulation in Python?

a) A class having private attributes
b) A class inheriting from multiple parent classes
c) A method having multiple implementations in different subclasses
d) None of the above

18. What is an attribute in Python?

a) A function that modifies the behavior of an object
b) A variable that holds a value associated with an object
c) A module that provides additional functionality to an object
d) None of the above

19. Which of the following is a valid way to access an attribute of an object in Python?

a) object.attribute_name
b) object[attribute_name]
c) Both A and B
d) None of the above

20. What is a method in Python?

a) A variable that holds a value associated with an object
b) A function that is associated with an object and can modify its state
c) A module that provides additional functionality to an object
d) None of the above

21. Which of the following is a valid way to call a method of an object in Python?

a) object.method_name
b) method_name(object)
c) Both A and B
d) None of the above

22. What is a getter in Python?

a) A method that retrieves the value of an attribute
b) A method that sets the value of an attribute
c) A method that modifies the behavior of an object
d) None of the above

23. Which of the following is a valid way to define a getter for an attribute in Python?

a) @ getter
b) @ property
c) @get
d) None of the above

24. What is a setter in Python?

a) A method that retrieves the value of an attribute
b) A method that sets the value of an attribute
c) A method that modifies the behavior of an object
d) None of the above

25. Which of the following is a valid way to define a setter for an attribute in Python?

a) @property
b) @setter
c) @set
d) None of the above

26. What is the purpose of using getters and setters in Python?

a) To provide a way to access and modify the state of an object
b) To encapsulate the implementation details of an object
c) To provide a way to define read-only and write-only attributes
d) All of the above

27. Which OOP concept allows a class to inherit properties and methods from multiple parent classes?

a) Multiple inheritance
b) Single inheritance
c) Multilevel inheritance
d) Hierarchical inheritance

28. What is the purpose of the "str" method in Python classes?

a) It returns a string representation of the object
b) It creates a new instance of the class
c) It initializes the object with default values
d) It compares two objects for equality

29. What is the output of the following code:

```
class MyClass:
    class_attribute = "Hello"

object = MyClass()
print(object.class_attribute)
```

a) "Hello"
b) MyClass
c) AttributeError: 'MyClass' object has no attribute 'class_attribute'
d) None of the above

30. What is the output of the following code:

```
class MyClass:
    def __init__(self):
        self.instance_attribute = "World"

object = MyClass()
print(object.instance_attribute)
```

a) "Hello"
b) "World"
c) MyClass
d) AttributeError: 'MyClass' object has no attribute 'instance_attribute

31. What is the output of the following code:

```
class MyClass:
    def __init__(self):
        self.instance_attribute = "Hello"

    def get_attribute(self):
        return self.instance_attribute

object = MyClass()
print(object.get_attribute())
```

a) "Hello"
b) MyClass
c) AttributeError: 'MyClass' object has no attribute 'get_attribute'
d) None of the above

32. What is the output of the following code:

```
class MyClass:
    def __init__(self):
        self.instance_attribute = "Hello"

    @property
    def attribute(self):
        return self.instance_attribute

object = MyClass()
print(object.attribute)
```

a) "Hello"
b) MyClass
c) AttributeError: 'MyClass' object has no attribute 'attribute'
d) None of the above

33. What is an abstract method in Python?

a) A method that must be overridden by any concrete subclass
b) A method that cannot be overridden by any concrete subclass
c) A method that can only be called from within the class that defines it
d) None of the above

34. What is the output of the following code:

```
class MyClass:
    def __init__(self):
        self.instance_attribute = "Hello"

    @attribute.setter
    def attribute(self, value):
        self.instance_attribute = value

object = MyClass()
object.attribute = "World"
print(object.attribute)
```

a) "Hello"
b) "World"
c) NameError: name 'attribute' is not defined
d) AttributeError: 'MyClass' object has no attribute 'attribute'

35. Which of the following statements about methods in Object-Oriented Programming (OOP) is true?

a) Methods define the state of an object
b) Methods are used to declare variables within a class
c) Methods can only be called from within the class where they are defined
d) Methods facilitate the behavior of objects and can manipulate their data

36. What is the output of the following code:

```
class MyClass:
    def __init__(self):
        self.__private_attribute = "Hello"

object = MyClass()
print(object.__private_attribute)
```

a) "Hello"
b) MyClass
c) AttributeError: 'MyClass' object has no attribute '__private_attribute'
d) None of the above

37. What is the purpose of using name mangling for private attributes in Python?
a) To prevent accidental modification of private attributes by other code
b) To enforce stricter encapsulation of the implementation details of an object
c) To make private attributes more easily accessible to other code
d) None of the above

38. Which of the following is a valid way to define a private method in Python?
a) def __method_name(self):
b) def _method_name(self):
c) def method_name(self):
d) None of the above

39. What is the output of the following code:

```
class MyClass:
    def __init__(self):
        self.__private_attribute = "Hello"

    def __private_method(self):
        return "World"

object = MyClass()
print(object._MyClass__private_attribute)
print(object._MyClass__private_method())
```

a) "Hello", "World"
b) AttributeError: 'MyClass' object has no attribute '_MyClass__private_attribute', "World"
c) "Hello", AttributeError: 'MyClass' object has no attribute '_MyClass__private_method'
d) AttributeError: 'MyClass' object has no attribute '_MyClass__private_attribute',
AttributeError: 'MyClass' object has no attribute '_MyClass__private_method'

40. What is the purpose of using inheritance in Python?

a) To define a new class based on an existing class
b) To reuse code and avoid duplication
c) To allow classes to be more easily compared to each other
d) All of the above

41. Can decorators be used to modify class methods in Python?

a) Yes, by decorating the method directly
b) No, decorators can only be used on functions
c) Yes, by decorating the class and its methods
d) None of the above

42. What is the purpose of a class decorator in Python?

a) To modify the behavior of all instances of a class
b) To add new methods or attributes to a class
c) To provide a convenient shorthand for creating instances of a class with a specific set of arguments
d) All of the above

43. What is the syntax for defining a class decorator in Python?

a) def decorator(cls):
b) @decorator
c) class decorator():
d) None of the above

44. Can decorators be used to modify instance methods in Python?

a) Yes, by decorating the method directly
b) No, decorators can only be used on class methods
c) Yes, by decorating the class and its methods
d) None of the above

45. What is the purpose of a property decorator in Python?

a) To add a new property to an existing class
b) To modify the behavior of an existing property of a class
c) To provide a convenient shorthand for accessing an attribute of a class
d) None of the above

46. What is the output of the following code:

```
class Wrapper:
    def __init__(self, **kwargs):
        self.__dict__.update(kwargs)

    def __getitem__(self, key):
        if len(self.__dict__) != 0:
            return self.__dict__[key]
        else:
            return "There is no such item"

obj = Wrapper(x=1, y=2, z=3)
obj.__dict__['x'] = 4
obj.__dict__['w'] = 5
print(obj['x'] + obj['y'] + obj['z'] + obj['w'])
```

a) 10
b) 12
c) 14
d) 15

47. Can decorators be used to modify class properties in Python?

a) Yes, by decorating the property directly
b) No, decorators can only be used on instance properties
c) Yes, by decorating the class and its properties
d) None of

48. What is the purpose of a class method in Python?

a) To modify the behavior of instances of a class
b) To create new instances of a class with a specific set of arguments
c) To add new methods or attributes to a class
d) All of the above

49. What is the difference between a class and a static methods in Python?

a) Class methods can only be called on instances of a class, while static methods can only be called on the class itself
b) Class methods receive the class as the first argument, while static methods do not receive any special arguments
c) Class methods are used for creating instances of a class, while static methods are used for modifying instances of a class
d) None of the above

50. What is the output of the following code:

```
class MyClass:
    def __init__(self, a):
        self._a = a

    @property
    def a(self):
        print("Getting attribute 'a'")
        return self._a

    @a.setter
    def a(self, value):
        print("Setting attribute 'a'")
        self._a = value

obj = MyClass(1)
obj.a = 2
print(obj.a)
```

a) "Setting attribute 'a'", "Getting attribute 'a'", "2"
b) "Getting attribute 'a'", "Setting attribute 'a'", "Getting attribute 'a'", "2"
c) "Getting attribute 'a'", "Setting attribute 'a'", "2"
d) None of the above

51. What is the syntax for defining a class method in Python?

a) def method(cls):
b) @method
c) class method():
d) None of the above

52. What is the output of the following code:

```
class MyClass:
    @classmethod
    def my_method(cls, a, b):
        print(f"a = {a}, b = {b}")

MyClass.my_method(1, 2)
```

a) "a = 1, b = 2"
b) "a = MyClass, b = 1, 2"
c) "a = 1, b = MyClass"
d) None of the above

53. What is the purpose of a static method in Python?

a) To modify the behavior of instances of a class
b) To create new instances of a class with a specific set of arguments
c) To add new methods or attributes to a class
d) None of the above

54. What is the purpose of an abstract method in Python?

a) To provide a template for subclasses to follow
b) To enforce a particular interface for a class hierarchy
c) To prevent certain methods from being called by instances of a class
d) All of the above

55. What is the output of the following code:

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

    def bark(self):
        print("Woof woof!")

class Labrador(Dog):
    def bark(self):
        print("Bark bark!")

dog1 = Labrador("Buddy")
dog1.bark()
```

a) Woof woof!
b) Bark bark!
c) Error
d) None of the above

56. What is the main difference between a class method and an instance method in Python?

a) Class methods can only be called on class instances, while instance methods can only be called on the class itself.
b) Class methods are defined with the @classmethod decorator, while instance methods are not.
c) Class methods operate on class-level data and can be called on both class instances and the class itself, while instance methods operate on instance-specific data and can only be called on class instances.
d) There is no difference between class methods and instance methods in Python.

57. What is the syntax for inheriting a class in Python?

a) class ChildClass(BaseClass()):
b) class ChildClass(BaseClass):
c) class ChildClass extends BaseClass
d) class BaseClass(ChildClass):

58. What is the output of the following code:

```
from abc import ABC, abstractmethod

class AbstractClass(ABC):
    @abstractmethod
    def some_method(self):
        print("It's AbstractClass Class")

class ConcreteClass(AbstractClass):
    def some_method(self):
        print("It's ConcreteClass Class.")

obj = AbstractClass()
obj.some_method()
```

a) It will print "It's AbstractClass Class".
b) It will print "It's ConcreteClass Class.".
c) It will raise a TypeError when calling obj.some_method().
d) It will raise a NameError because the method is undefined.

59. Which operator overload dunder method is used to implement the //= operator in Python?

a) __ifloordiv__
b) __floordiv__
c) __rfloordiv__
d) __tfloordiv__

60. Which dunder method is used to implement the del statement in Python?

a) __del__
b) __delete__
c) __clear__
d) __remove__

61. Which of the following is a valid use of the __getattr__ dunder method?

a) To define the behavior of the == operator.
b) To define the behavior of the [] operator.
c) To define the behavior of undefined attributes.
d) To define the behavior of the + operator.

62. **Which operator overload dunder method is used to implement the != operator in Python?**

a) __eq__
b) __ne__
c) __lt__
d) __le__

63. **What is the output of the following code:**

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

    def __str__(self):
        return str(self.value)

a = MyClass(5)
b = MyClass(10)

print(a + b)
```

a) TypeError
b) 15
c) "15"
d) "MyClass(15)"

64. **What is the output of the following code:**

```python
class MyClass:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __str__(self):
        return f"MyClass(a={self.a}, b={self.b})"

class MyChildClass(MyClass):
    def __init__(self, a, b, c):
        super().__init__(a, b)
        self.c = c

    def __str__(self):
        return f"MyChildClass(a={self.a}, b={self.b}, c={self.c})"

    @staticmethod
    def my_method():
        return MyClass(1, 2)
print(MyChildClass.my_method())
```

a) "MyChildClass(a=1, b=2, c=None)"
b) "MyClass(a=1, b=2)"
c) "MyChildClass(a=1, b=2, c=MyClass(a=1, b=2))"
d) None of the above

65. What is the output of the following code:

```
class MyClass:
    x = 0

    def __init__(self, a, b):
        self.a = a
        self.b = b

    @classmethod
    def my_method(cls, c):
        cls.x += c
        return cls.x

obj1 = MyClass(1, 2)
obj2 = MyClass(3, 4)
print(obj1.my_method(1))
print(obj2.my_method(2))
```

a) 1, 2
b) 2, 3
c) 1, 3
d) 2, 4

66. What is the method used to call a method in a parent class from a child class?
a) parent_method()
b) super()
c) child_method()
d) base()

67. Which of the following is a disadvantage of using inheritance?
a) It can make the code more readable and organized
b) It can make the code more complex and harder to maintain
c) It can only be used with built-in Python classes
d) It can only be used with single inheritance, not multiple inheritance

68. What is the output of the following code:

```
from abc import ABC, abstractmethod

class MyAbstractClass(ABC):
    @abstractmethod
    def my_method(self):
        pass

class MyClass(MyAbstractClass):
    def my_method(self):
        print("Hello, world!")

class MyChildClass(MyClass):
    @staticmethod
    def my_method():
        return "Static method"

obj = MyChildClass()
print(obj.my_method())
```

a) "Hello, world!"
b) "Static method"
c) TypeError: my_method() missing 1 required positional argument: 'self'
d) None of the above

69. What is the output of the following code:

```
class MyClass:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    @staticmethod
    def my_method(a, b):
        return a + b

class MyChildClass(MyClass):
    def __init__(self, a, b, c):
        super().__init__(a, b)
        self.c = c
    @property
    def my_method(self):
        return self.a + self.b + self.c

obj = MyChildClass(1, 2, 3)
print(obj.my_method())
```

a) 8
b) 6
c) TypeError: 'MyClass' object is not callable
d) None of the above

70. What is the output of the following code:

```
class MyClass:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def my_method(self):
        return str(self.a) + str(self.b)

    @classmethod
    def my_classmethod(cls, a, b):
        return cls(a, b)

obj = MyClass.my_classmethod(1, 2)
print(obj.my_method())
```

a) 3
b) 12
c) TypeError: 'MyClass' object is not callable
d) None of the above

71. What is the output of the following code:

```
class MyClass:
    def __init__(self, value):
        self.__value = value

    def get_value(self):
        return self.__value

obj = MyClass(10)
print(obj.get_value())
print(obj.__value)
```

a) 10, 10
b) 10, AttributeError
c) AttributeError, AttributeError
d) AttributeError, 10

72. **Which of the following statements is true about inheritance?**

a) A child class can access all the properties and methods of its parent class
b) A parent class can access all the properties and methods of its child class
c) A child class can only access the public properties and methods of its parent class
d) A child class can access the private properties and methods of its parent class

73. **What is hierarchical inheritance in Python?**

a) Hierarchical inheritance is a type of inheritance where a class inherits from multiple parent classes.
b) Hierarchical inheritance is a type of inheritance where a class inherits from a single parent class.
c) Hierarchical inheritance is a type of inheritance where a class inherits from its own derived classes.
d) Hierarchical inheritance is not supported in Python.

74. **What is the output of the following code:**

```
class MyClass:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def my_method(self):
        return self.a + self.b

class MyChildClass(MyClass):
    def __init__(self, a, b, c):
        super().__init__(a, b)
        self.c = c

    def my_method(self):
        return super().my_method() + self.c

obj = MyChildClass(1, 2, 3)
print(obj.my_method())
```

a) 6
b) 9
c) TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
d) None of the above

75. What is the output of the following code:

```
class MyClass:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __eq__(self, other):
        return self.a == other.a and self.b == other.b

a = MyClass(1, 2)
b = MyClass(1, 3)

print(a == b)
```

a) True
b) False
c) TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
d) None of the above

76. What is the order of method resolution in Python for multiple inheritance?

a) Depth-first, left-to-right
b) Breadth-first, right-to-left
c) Depth-first, right-to-left
d) Breadth-first, left-to-right

77. What happens if there is a conflict in method names between parent classes during MRO?

a) An error is raised, indicating a method name conflict
b) The method from the first parent class defined in the inheritance list is chosen
c) The method from the last parent class defined in the inheritance list is chosen
d) The programmer needs to explicitly specify which method to use by overriding the method in the child class

78. How is the Method Resolution Order (MRO) determined in Python?

a) It is determined based on the order in which the classes are defined
b) It is determined based on the inheritance hierarchy specified using the super() function
c) It is determined based on the linearization of the class hierarchy using the C3 algorithm
d) It is determined randomly at runtime

79. What is the output of the following code:

```
class MyClass:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def my_method(self):
        return self.a + self.b

    @staticmethod
    def my_staticmethod(a, b):
        return a + b

    @classmethod
    def my_classmethod(cls, a, b):
        return cls(a, b)

class MyChildClass(MyClass):
    def __init__(self, a, b, c):
        super().__init__(a, b)
        self.c = c

    @staticmethod
    def my_staticmethod(a, b, c):
        return a + b + c

    @classmethod
    def my_classmethod(cls, a, b, c):
        return cls(a, b, c)

obj1 = MyClass.my_classmethod(1, 2)
obj2 = MyChildClass.my_classmethod(1, 2, 3)

print(obj1.my_staticmethod(3, 4))
print(obj2.my_staticmethod(3, 4, 5))
```

a) **3, 6**
b) **TypeError: my_staticmethod() takes 2 positional arguments but 3 were given**
c) **7, 12**
d) **None of the above**

80. What is the output of the following code:

```
class ParentClass:
    def __init__(self, a):
        self.a = a

class ChildClass(ParentClass):
    def __init__(self, a=1, b=5):
        super().__init__(a)
        self.b = b

obj = ChildClass(a=2)
print(obj.a, obj.b)
```

a) 1, 5
b) 2, 5
c) TypeError: init() missing 1 required positional argument: 'b'
d) AttributeError: 'ChildClass' object has no attribute 'a'

81. What is the output of the following code:

```
class ParentClass:
    def __init__(self, a):
        self.a = a

class ChildClass(ParentClass):
    def __init__(self, a, b):
        super().__init__(a)
        self.b = b

class GrandChildClass(ChildClass):
    def __init__(self, a, b, c):
        super().__init__(a, b)
        self.c = c

obj = GrandChildClass(6%4, 2+1, 3//2)
print(obj.a, obj.b, obj.c)
```

a) 1, 2, 3
b) 1, 3, 2
c) 2, 1, 3
d) 2, 3, 1

82. Which of the following statements best describes encapsulation in object-oriented programming?

a) It is the ability of an object to take on many forms.
b) It is the process of hiding the implementation details of an object from the outside world.
c) It is the process of creating a new class from an existing class.
d) It is the ability of a class to inherit properties and behaviors from another class.

83. Which of the following statements best describes polymorphism in object-oriented programming?

a) It is the process of hiding the implementation details of an object from the outside world.
b) It is the process of creating a new class from an existing class.
c) It is the ability of an object to take on many forms.
d) It is the ability of a class to inherit properties and behaviors from another class.

84. What is the output of the following code:

```
class ParentClass:
    def my_method(self):
        print("Child method")

class ChildClass(ParentClass):
    def my_method(self):
        print("Parent method")

def call_method(obj):
    obj.my_method()

parent_obj = ParentClass()
child_obj = ChildClass()

call_method(parent_obj)
call_method(child_obj)
```

a) "Parent method", "Child method"
b) "Child method", "Parent method"
c) "Parent method", "Parent method"
d) "Child method", "Child method"

85. Which of the following is an example of duck typing in Python?

a) Using type hints in a function definition to specify the expected input types.
b) Using the isinstance() function to check the type of an object before using it.
c) Using try-except blocks to handle different types of exceptions that may arise during program execution.
d) Writing a function that works with any object that has a certain set of methods, regardless of its type.

86. What is the difference between method overloading and method overriding in Python?

a) Method overloading refers to defining multiple methods with the same name in a class, while method overriding refers to redefining a method in a subclass that was already defined in the parent class.
b) Method overriding refers to defining multiple methods with the same name in a class, while method overloading refers to redefining a method in a subclass that was already defined in the parent class.
c) Method overloading and method overriding are the same thing in Python.
d) None of the above.

87. What is the output of the following code:

```
class Shape:
    def area(self):
        return 0

class Square(Shape):
    def __init__(self, length):
        self.__length = length

    def area(self):
        return self.__length ** 2

class Circle(Shape):
    def __init__(self, radius):
        self.__radius = radius

    def area(self):
        return 3.14 * self.__radius ** 2

shapes = [Square(5), Circle(3)]
for shape in shapes:
    print(shape.area())
```

a) 25, 9.42
b) 25, 38.26
c) 25, 28.26
d) SyntaxError

88. Which of the following is not a type of inheritance in Python?

a) Single inheritance
b) Multiple inheritance
c) Hierarchical inheritance
d) Circular inheritance

89. What is the output of the following code:

```
from abc import ABC, abstractmethod

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

class Square(Shape):
    def __init__(self, length):
        self.__length = length

    def area(self):
        return self.__length ** 2

square = Square(5, 5)
print(square.area())
```

a) 25
b) 0
c) TypeError
d) AttributeError

90. What is the purpose of magic methods in Python?
a) To perform mathematical operations.
b) To define special behaviours for built-in operators and functions.
c) To define private methods in a class.
d) To generate random numbers.

91. Which of the following is a correct syntax for a magic method in Python?
a) __method__(self, other)
b) method__(self, other)
c) __method(self, other)__
d) method__(self, other)__

92. Which of the following magic methods is used to define the behaviour of the addition operator (+) in Python?
a) __add__
b) __init__
c) __str__
d) __repr__

93. What is the output of the following code:

```
class Point:
    def __init__(self, x=0, y=0):
        self.__x = x
        self.__y = y

    def __str__(self):
        return f"({self.__x}, {self.__y})"

    def __add__(self, other):
        return Point(self.__x + other.__x, self.__y + other.__y)

p1 = Point(-2, -2)
p2 = Point(5, 3)
p3 = p1 + p2
print(p3)
```

a) (1, 5)
b) (3, 1)
c) (2, 3, -1, 2)
d) TypeError

94. Which of the following magic methods is used to define the behaviour of the equality operator (==) in Python?

a) __init__
b) __eq__
c) __str__
d) __repr__

95. What is the output of the following code:

```
class A:
    def __init__(self, x):
        self.__x = x

    def __add__(self, other):
        return self.__x + other.__x

a = A(2)
b = A(3)
c = pow(a+b, 2)
print(c)
```

a) 2
b) 3
c) 5
d) 25

96. What is the output of the following code:

```
class Rectangle:
    def __init__(self, length, width):
        self.__length = length
        self.__width = width

    def __str__(self):
        return f"Rectangle({self.__length}, {self.__width})"

    def __eq__(self, other):
        return self.__length == other.__length and self.__width ==
other.__width

r1 = Rectangle(2, 3)
r2 = Rectangle(3, 2)
r3 = Rectangle(2, 3)
print(r1 == r2)
print(r1 == r3)
```

a) True, False
b) False, True
c) True, True
d) False, False

97. What is the output of the following code:

```
class Student:
    def __init__(self, name, age):
        self.__name = name
        self._age = age

    def __repr__(self):
        return f"Student({self.__name}, {self._age})"

students = [Student("Alice", 20), Student("Bob", 22), Student("Charlie", 18)]
print(sorted(students, key=lambda s: s._age))
```

a) Student(Bob, 22), Student(Alice, 20), Student(Charlie, 18)
b) Student(Alice, 20), Student(Bob, 22), Student(Charlie, 18)
c) Student(Charlie, 18), Student(Alice, 20), Student(Bob, 22)
d) AttributeError

98. **What is the output of the following code:**

```
class Vehicle:
    def __init__(self, speed):
        self.speed = speed

    def __str__(self):
        return "Vehicle"

class Car(Vehicle):
    def __init__(self, speed, model):
        super().__init__(speed)
        self.model = model

    def __str__(self):
        return f"{super().__str__()} - Car - {self.model}"

car1 = Car(100, "BMW")
print(car1)
```

a) BMW - Car - Vehicle
b) Car - BMW
c) Vehicle - Car - BMW
d) BMW – Car

99. **What is the output of the following code:**

```
class A:
    pass

class B(A):
    pass

class C(B):
    pass

print(C.mro())
```

a) [<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <class 'object'>]
b) [<class '__main__.C'>, <class '__main__.B'>, <class '__main__.A'>, <class 'object'>]
c) [<class '__main__.C'>, <class '__main__.B'>, <class 'object'>, <class '__main__.A'>]
d) [<class '__main__.C'>, <class '__main__.B'>, <class '__main__.A'>, <class 'object'>]

100. **What is the output of the following code:**

```
class A:
    def __init__(self):
        self.x = 7

class B(A):
    def __init__(self):
        super().__init__()
        self.y = 0

class C(A):
    def __init__(self):
        super().__init__()
        self.z = 2

class D(B, C):
    def __init__(self):
        super().__init__()

d = D()
print(d.x + d.y + d.z)
```

a) 7
b) 9
c) 2
d) AttributeError

101. **What is the output of the following code:**

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

    def __mul__(self, other):
        return A(self.x * other.x)

a1 = A(5)
a2 = A(3)
a3 = a1 * a2
print(a3.x)
```

a) 3
b) 5
c) 15
d) TypeError

102. **What is the output of the following code:**

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

class B(A):
    def __init__(self, x, y):
        A.__init__(self, x)
        self.y = y

class C(B):
    def __init__(self, x, y, z):
        B.__init__(self, x, y)
        self.z = z

c = C(1, 2, 3)
print(c.x + c.u + c.z)
```

a) 6
b) 7
c) 8
d) AttributeError

103. **What is the output of the following code:**

```
class A:
    pass

class B(A):
    pass

class C(B):
    pass

class D:
    pass

obj = C()
print(isinstance(obj, D))
```

a) True
b) False
c) Error
d) None of the above

104. **What is the output of the following code:**

```
class A:
    def method(self):
        print("A method")

class B(A):
    def method(self):
        print("B method")

class C(A):
    def method(self):
        print("C method")

class D(B, C):
    def method(self):
        super().method()

class E(D, A):
    pass

e = E()
e.method()
```

a) A method
b) B method
c) C method
d) TypeError

105. **Which of the following is not a benefit of using OOP in Python?**

a) Code reusability
b) Encapsulation of data and behavior
c) Improved performance
d) Modularity and maintainability

106. **What is the output of the following code:**

```
class Furniture:
    def __init__(self, material):
        self.material = material

    def display(self):
        print("Material:", self.material)

class Chair(Furniture):
    def __init__(self, material, legs):
        super().__init__(material)
        self.legs = legs

    def display(self):
        super().display()
        print("Legs:", self.legs)

chair = Chair("Wood", 4)
chair.display()
```

a) Material: Wood, Legs: 4
b) Material: Wood
c) Compilation error
d) Runtime exception

107. **What is the output of the following code:**

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

    def __eq__(self, other):
        return self.x == other.x

class B(A):
    def __init__(self, x, y):
        super().__init__(x)
        self.y = y

    def __eq__(self, other):
        return super().__eq__(other) and self.y == other.y

a = A(2)
b = B(2, 3)
print(a == b)
```

a) True
b) False
c) TypeError
d) AttributeError

108. What is the output of the following code:

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

class B(A):
    def __init__(self, x, y):
        A.__init__(self, x)
        self.y = y

a = A(1)
b = B(2, 3)
print(a.x + b.x + b.y)
```

a) 6
b) 5
c) 4
d) AttributeError

109. What is the output of the following code:

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

    def __repr__(self):
        return f"A({self.x})"

class B(A):
    def __init__(self, x, y):
        super().__init__(x)
        self.y = y

    def __repr__(self):
        return f"B({self.x}, {self.y})"

b = B(2, 3)
print(b)
```

a) A(2)
b) B(2, 3)
c) A object at memory location 0x...
d) B object at memory location 0x...

110. What is the output of the following code:

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

    def __add__(self, other):
        return A(self.x + other.x)

a1 = A(2)
a2 = A(6)
a3 = a1 + a2
print(a3.x)
```

a) 5
b) 6
c) 8
d) TypeError

111. What is the output of the following code:

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

class B(A):
    def __init__(self, x, y):
        A.__init__(self, x)
        self.y = y

    def __str__(self):
        return f"{self.x}, {self.y}"

a = B(1, 2)
print(b)
```

a) 1, 2
b) NameError
c) AttributeError
d) TypeError

112. **What is the output of the following code:**

```
class B():
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return f"{self.x}, {self.y}"

class C(B):
    def __init__(self, x, y, z):
        B.__init__(x, y)
        self.z = z

    def __str__(self):
        return f"{self.x}, {self.y}, {self.z}"

c = C(1, 2, 3)
print(c)
```

a) **1, 2, 3**
b) **C object at memory location 0x...**
c) **AttributeError**
d) **TypeError**

113. **What is the output of the following code:**

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

    def __repr__(self):
        return f"A({self.x})"

class B(A):
    def __init__(self, x, y):
        A.__init__(self, x)
        self.y = y

    def __repr__(self):
        return f"B({self.x}, {self.y})"

b = B(1, 2)
print([b])
```

a) [A(1), 2]
b) [B(1, 2)]
c) TypeError
d) AttributeError

114. What is the output of the following code:

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

    def __add__(self, other):
        if self.x < 2:
            self.x += 1
            self.y = 0
        return A(self.x + other.x)

class B(A):
    def __init__(self, x, y):
        A.__init__(self, x)
        self.y = y

    def __add__(self, other):
        if self.x < 2:
            self.x += 11
            self.y -= 2
        return B(self.x + other.x, self.y + other.y)

a1 = A(1)
a2 = A(2)
a3 = a1 + a2
print(a3.x)

b1 = B(1, 2)
b2 = B(3, 4)
b3 = b1 + b2
print(b3.x + b3.y)
```

a) 4, 10
b) 3, 5
c) 3, 10
d) 4, 19

115. What is the output of the following code:

```
class Building:
    def __init__(self, name, location):
        self.name = name
        self.location = location

class Skyscraper(Building):
    def __init__(self, name, location, floors):
        super().__init__(name, location)
        self.floors = floors

    def print_details(self):
        print(f"This is {self.name}, located in {self.location}, and has
{self.floors} floors.")

empire_state = Skyscraper("Empire State Building", "New York City", 102)
empire_state.print_details()
```

a) This is Empire State Building, located in New York City, and has 102 floors.
b) This is Empire State Building, and has 102 floors.
c) This is Skyscraper, located in New York City, and has 102 floors.
d) This code will produce a syntax error.

116. What is the output of the following code:

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def __str__(self):
        return f'{self.year} {self.make} {self.model}'

class ElectricCar(Car):
    def __init__(self, make, model, year, battery_size):
        super().__init__(make, model, year)
        self.battery_size = battery_size

    def __str__(self):
        return f'{self.year} {self.make} {self.model} with a
{self.battery_size}-kWh battery'

my_tesla = ElectricCar('tesla', 'model s', 2022, 75)
print(my_tesla)
```

a) 'tesla model s 2022'
b) '2022 tesla model s'
c) '2022 tesla model s with a 75-kWh battery'
d) 'tesla model s 2022 with a 75-kWh battery'

117. **What is the output of the following code:**

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

    def __str__(self):
        return f"A: {self.x}"

class B:
    def __init__(self, y):
        self.y = y

    def __str__(self):
        return f"B: {self.y}"

class C(A, B):
    def __init__(self, x, y, z):
        A.__init__(self, x)
        B.__init__(self, y)
        self.z = z

    def __str__(self):
        return f"{A.__str__(self)}, {B.__str__(self)}, C: {self.z}"

c = C(1, 2, 3)
print(c)
```

a) A: 1, B: 2, C: 3
b) C: 3
c) TypeError
d) AttributeError

118. **What is the output of the following code?**

```
class MyDict:
    def __init__(self, d):
        self.d = d
    def __getitem__(self, key):
        return self.d[key]
    def __setitem__(self, key, value):
        self.d[key] = value
    def __repr__(self):
        return str(self.d)

a = MyDict({'a': 1, 'b': 2})
a['c'] = 3
a["a"] = 4
print(a)
```

a) {'a': 1, 'b': 2, 'c': 3}
b) {'a': 1, 'b': 2}
c) {'a': 4, 'b': 2, 'c': 3}
d) TypeError: 'MyDict' object does not support item deletion

119. **What is the output of the following code:**

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

class ElectricCar(Car):
    def __init__(self, make, model, year, battery_size):
        super().__init__(make, model, year)
        self.battery_size = battery_size

my_tesla = ElectricCar('tesla', 'model s', 2022, 75)
print(my_tesla.battery_size)
```

a) 'tesla'
b) 'model s'
c) 2022
d) 75

120. **What is the output of the following code?**

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

class Dog(Animal):
    def __init__(self, name):
        super().__init__(name)

    def speak(self):
        return 'woof!'

my_dog = Dog('Fido')
print(my_dog.name.lower())
print(my_dog.speak())
```

a) 'Fido' 'woof!'
b) 'fido' 'woof!'
c) None 'woof!'
d) None None

121. **What is the output of the following code:**

```
class Device:
    def __init__(self, name, brand):
        self.name = name
        self.brand = brand

    def turn_on(self):
        print(f"{self.name} is turning on.")

class Smartphone(Device):
    def __init__(self, name, brand, os):
        super().__init__(name, brand)
        self.os = os

    def turn_on(self):
        print(f"{self.name} running {self.os} is turning on.")

iphone = Smartphone("iPhone", "Apple", "iOS")
samsung = Smartphone("Galaxy", "Samsung", "Android")

devices = [iphone, samsung]
for device in devices:
    device.turn_on()
```

a) iPhone is turning on, Galaxy is turning on.
b) iPhone running iOS is turning on, Galaxy running Android is turning on.
c) This code will produce a syntax error.
d) None of the above.

122. What is the output of the following code:

```
class BankAccount:
    def __init__(self, name, balance):
        self.name = name
        self.balance = balance

    def __str__(self):
        return f"{self.name}'s account has a balance of {self.balance}
dollars."

    def __add__(self, other):
        return BankAccount(f"{self.name} and {other.name}", self.balance +
other.balance)

    def __sub__(self, amount):
        if amount > self.balance:
            return f"Error: {self.name}'s account has insufficient funds."
        self.balance -= amount
        return f"{self.name} withdrew {amount} dollars. New balance is
{self.balance} dollars."

    def __rsub__(self, amount):
        return self.__sub__(amount)

    def __mul__(self, other):
        return BankAccount(f"{self.name} and {other.name} joint account",
self.balance + other.balance)

    def __rmul__(self, other):
        return self.__mul__(other)

    def __eq__(self, other):
        return self.balance == other.balance

account1 = BankAccount("Alice", 500)
account2 = BankAccount("Bob", 1000)
joint_account = (account1 * account2) + account1

print(joint_account)
```

a) Alice withdrew 200 dollars. New balance is 300 dollars.I am Bob and I am 40 years old. I manage a team of 10 clerks
b) Bob withdrew 500 dollars. New balance is 500 dollars.None of the above
c) Alice and Bob joint account and Alice's account has a balance of 2000 dollars
d) Alice and Bob joint account's account has a balance of 1500 dollars

123. What is the output of the following code:

```
class Clerk:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def print_info(self):
        print(f"I am {self.name} and I am {self.age} years old. I work as a
{type(self).__name__}.")


class Cashier(Clerk):
    def __init__(self, name, age, salary):
        super().__init__(name, age)
        self.salary = salary

    def print_info(self):
        print(f"I am {self.name} and I am {self.age} years old. I work as a
{type(self).__name__} and my salary is {self.salary}.")


class Manager(Clerk):
    def __init__(self, name, age, team_size):
        super().__init__(name, age)
        self.team_size = team_size

    def print_info(self):
        print(f"I am {self.name} and I am {self.age} years old. I work as a
{type(self).__name__} and I manage a team of {self.team_size} clerks.")


clerk1 = Clerk("John", 25)
cashier1 = Cashier("Alice", 30, 4000)
manager1 = Manager("Bob", 40, 10)

clerks = [clerk1, cashier1, manager1]

clerks[1].print_info()
```

a) I am John and I am 25 years
b) I am Bob and I am 40 years old. I manage a team of 10 clerks
c) I am Alice and I am 30 years old. I work as a <class '__main__.Cashier'> and my salary is 4000.
d) I am Alice and I am 30 years old. I work as a Cashier My salary is 4000.

124. What is the output of the following code:

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def __str__(self):
        return f"{self.make} {self.model} ({self.year})"

    def __repr__(self):
        return f"{self.make} {self.model} ({self.year})"

class ElectricCar(Car):
    def __init__(self, make, model, year, battery_size):
        super().__init__(make, model, year)
        self.battery_size = battery_size

    def get_range(self):
        if self.battery_size == 75:
            return 260
        elif self.battery_size == 100:
            return 315

my_car = ElectricCar('Tesla', 'Model S', 2022, 75)
print(my_car)
```

a) Tesla Model S (2022)
b) Tesla Model S (2022) 260
c) Tesla Model S (2022) 315
d) ElectricCar object at 0x000001

125. What is the output of the following Python code?

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

```
    def __mul__(self, other):
        return self.x * other.x

a = MyNumber(5)
b = MyNumber(2)

print(a * b * 2)
```

a) 10
b) 20
c) 100
d) 200

126. What is the output of the following code:

```
class Animal:
    def __init__(self, species, weight):
        self.species = species
        self.weight = weight

class Dog(Animal):
    def __init__(self, breed, weight):
        super().__init__('Canine', weight)
        self.breed = breed

    def __str__(self):
        return f"{self.breed} ({self.species}) weighing {self.weight} pounds."

my_dog = Dog('Golden Retriever', 60)
print(my_dog)
```

a) Canine (Golden Retriever) weighing 60 pounds.
b) Dog (Golden Retriever) weighing 60 pounds.
c) Golden Retriever (Canine) weighing 60 pounds.
d) Animal (Golden Retriever) weighing 60 pounds.

127. **What is the output of the following code:**

```
class Vehicle:
    def __init__(self, brand):
        self.brand = brand

    def start(self):
        print("Starting the vehicle...")

class Car(Vehicle):
    def __init__(self, brand, color):
        super().__init__(brand)
        self.color = color

    def start(self):
        super().start()
        print("Starting the car...")

c = Car("Honda", "red")
c.start()
```

a) Starting the car...
b) Starting the vehicle...
c) Starting the vehicle... Starting the car...
d) Starting the car... Starting the vehicle...

128. **What is the output of the following code:**

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

    def greet(self):
        print(f"Hello, my name is {type(self).__name__} and I'm {self.age}
years old.")

class Employee(Person):
    def __init__(self, name, age, job_title):
        super().__init__(name, age)
        self.job_title = job_title

    def greet(self):
        super().greet()
        print(f"I work as a {self.job_title}.")
```

```
my_employee = Employee('John', 30, 'Software Engineer')
my_employee.greet()
```

a) Hello, my name is John and I'm 30 years old. I work as a Software Engineer.
b) I work as a Software Engineer. Hello, my name is John and I'm 30 years old.
c) Hello, my name is Employee and I'm 30 years old. I work as a Software Engineer.
d) I work as a Software Engineer.

129. What is the output of the following code:

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

class MuslimPerson(Muslim):
    def __init__(self, name):
        super().__init__(name)
    def greet(self):
        return f"Assalamu Alaikum, I am {self.name}"

person = MuslimPerson("Ahmed")
print(person.greet())
```

a) Assalamu Alaikum, I am Ahmed
b) Assalamu Alaikum
c) Ahmed
d) None

130. What will be the output of the following code:

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

    def description(self):
        return f"Islam is a religion called {self.name}"

class Muslim(Islam):
    def __init__(self, name, sect):
        super().__init__(name)
        self.sect = sect

muslim = Muslim("Ahmed", "Sunni")
print(muslim.description())
```

a) Islam is a religion called Sunni
b) Islam is a religion called Ahmed
c) Islam
d) None

131. What will be the output of the following code snippet?

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

    def location(self):
        return f"The airport {self.name} is located in {self.country}"

class InternationalAirport(Airport):
    def __init__(self, name, country):
        super().__init__(name)
        self.country = country

airport = InternationalAirport("Heathrow", "United Kingdom")
print(airport.location())
```

a) The airport Heathrow is located in United Kingdom
b) The airport is located in United Kingdom
c) Heathrow
d) None

132. What will be the output of the following code snippet?

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

    def start(self):
        return f"The war {self.name} has started in {self.year}"

class WorldWar(War):
    def __init__(self, name, year):
        super().__init__(name)
        self.year = year

war = WorldWar("WWII", 1945)
print(war.start())
```

a) The war WWII has started
b) The war has started
c) The war WWII has started in 1945
d) AttributeError: 'War' object has no attribute 'year'

133. What will be the output of the following code snippet?

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

    def job_description(self):
        return f"I am a {self.title} working in the field of {self.field}"

class Developer(Jobs):
    def __init__(self, title, programming_language):
        super().__init__(title)
        self.programming_language = programming_language
        self.field = "Software Engineering"

developer = Developer("Software Engineer", "Python")
print(developer.job_description())
```

a) I am a Software Engineer working in the field of Python
b) I am a Software Engineer working in the field of Software Engineering
c) Software Engineer
d) None

134. What will be the output of the following code snippet?

```
class LinkedIn:
    def __init__(self, username):
        self.username = username

class LinkedInUser(LinkedIn):
    def __init__(self, username, connections):
        super().__init__(username)
        self.connections = connections

user = LinkedInUser("JohnDoe", 500)
print(f"{user.username} has {user.connections} connections")
```

a) JohnDoe
b) LinkedIn
c) JohnDoe has 500 connections
d) Error

135. What will be the output of the following code snippet?

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

class Fruit(Food):
    def __init__(self, name, color):
        super().__init__(name)
        self.color = color

class Apple(Fruit):
    def __init__(self, name, color, variety):
        super().__init__(name, color)
        self.variety = variety

apple = Apple("Granny Smith", "Sour", "Green")
print(apple.color)
```

a) Green
b) Sour
c) Fruit
d) None

136. What will be the output of the following code snippet?

```
class Accessories:
    def __init__(self, category):
        self.category = category

class Necklace(Accessories):
    def __init__(self, category, material):
        super().__init__(category)
        self. category = material


necklace = Necklace("Jewelry", "Gold")
print(necklace.category)
```

a) Jewelry
b) Accessories
c) Gold
d) Error

137. What will be the output of the following code snippet?

```
class Sweat:
    def __init__(self, intensity):
        self.intensity = intensity

class Exercise(Sweat):
    def __init__(self, intensity, duration):
        super().__init__(intensity)
        self.duration = duration

class Cardio(Exercise):
    def __init__(self, intensity, duration, equipment):
        super().__init__(intensity, duration)
        self.equipment = equipment

cardio = Cardio("High", 30, "Treadmill")
print(cardio.intensity)
```

a) High
b) Sweat
c) None
d) Error

138. What will be the output of the following code snippet?

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

    def display(self):
        return f"Displaying {self.country} Climate Dashboard."

class SalesDashboard(Dashboard):
    def __init__(self, name, sales_data):
        super().__init__(name)
        self.sales_data = sales_data
        self.total = sum(self.sales_data)
```

```
    def display(self):
        return f"Displaying {self.name} dashboard with total {self.total}."

class ClimateDashboard(Dashboard):
    def __init__(self, country):
        self.country = country

dashboard = Dashboard("General")
sales_dashboard = SalesDashboard("Sales", [100, 200, 300, 400])
climate_dashboard = ClimateDashboard("Egypt")

print(sales_dashboard.display())
print(climate_dashboard.display())
```

a) **Displaying General dashboard.**
   Displaying Sales dashboard with total 1000.
b) **Displaying Sales dashboard with total 1000.**
   Displaying Egypt Climate Dashboard.
c) **AttributeError: 'ClimateDashboard' object has no attribute 'display'**
d) **AttributeError: 'Dashboard' object has no attribute 'country'**

139. What will be the output of the following code snippet?

```
class Course:
    def __init__(self, name, credits, grade):
        self.name = name
        self.credits = credits
        self.grade = grade

class Transcript:
    def __init__(self, courses):
        self.courses = courses

    def calculate_gpa(self):
        total_credits = 0
        total_grade_points = 0

        for course in self.courses:
            total_credits += course.credits
            total_grade_points += course.credits * course.grade

        gpa = total_grade_points / total_credits
        return round(gpa, 2)
```

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

    def check_gpa(self):
        gpa = self.transcript.calculate_gpa()

        if gpa >= 3.5:
            return f"Great job, {self.name}! Your GPA is {gpa:.2f}. Keep up the
excellent work!"
        elif gpa >= 2.0:
            return f"Good job, {self.name}! Your GPA is {gpa:.2f}. Keep
striving for improvement!"
        else:
            return f"Warning, {self.name}! Your GPA is {gpa:.2f}. Please work
harder to improve your grades."


# Example usage
courses = [
    Course("Math", 3, 3.7),
    Course("Science", 4, 3.2),
    Course("History", 3, 3.8),
    Course("English", 3, 2.5),
    Course("Art", 2, 4.0)]

transcript = Transcript(courses)
student = Student("John", transcript)
message = student.check_gpa()

print(message)
```

a) **Great job, John! Your GPA is 3.50. Keep up the excellent work!**
b) **Good job, John! Your GPA is 3.39. Keep striving for improvement!**
c) **Warning, John! Your GPA is 1.38. Please work harder to improve your grades**
d) **None of the above**

140. What will be the output of the following code snippet?

```
class Vehicle:
    def __init__(self, brand):
        self.brand = brand

    def start(self):
        return f"Starting {self.brand} vehicle."

class Motorcycle(Vehicle):
    def __init__(self, brand, model):
        super().__init__(brand)
        self.model = model

    def start(self):
        return f"Starting {self.brand} {self.model} motorcycle."

vehicle = Vehicle("Car")
motorcycle = Motorcycle("Honda", "CBR")

print(vehicle.start())
print(motorcycle.start())
```

a) **Starting Car vehicle.**
   Starting Honda CBR motorcycle.
b) **Car**
   Honda CBR
c) **Error**
d) **None of the above**

141. What will be the output of the following code:

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

    def boot(self):
        return f"Booting {self.name} OS."

class Linux(OperatingSystem):
    def __init__(self, name, version):
        super().__init__(name)
        self.version = version

    def boot(self):
        return f"Booting {self.name} {self.version} OS."

class Windows(OperatingSystem):
    def __init__(self, name, version):
        super().__init__(name)
        self.version = version

    def boot(self):
        return f"Booting {self.name} {self.version} OS."

class Mac(OperatingSystem):
    def __init__(self, name, version):
        super().__init__(name)
        self.version = version

    def boot(self):
        return f"Booting {self.name} {self.version} OS."

os = OperatingSystem("Windows")
linux = Linux("Ubuntu", "20.04")
windows = Windows("Windows", "10")
mac = Mac("macOS", "Catalina")

print(linux.boot())
```

a) **Booting Windows OS.**
b) **Booting Ubuntu 20.04 OS.**
c) **Booting Windows 10 OS.**
d) **Booting macOS Catalina OS.**

142. What will be the output of the following code snippet?

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

    def simulate(self):
        return f"Simulating {self.name} logic design."

class Adder(LogicDesign):
    def __init__(self, name, bit_width):
        super().__init__(name)
        self.bit_width = bit_width

    def simulate(self):
        return f"Simulating {self.name} {self.bit_width}-bit adder."

design = LogicDesign("Flip-Flop")
adder_design = Adder("Ripple Carry", 8)

print(design.simulate())
print(adder_design.simulate())
```

a) Simulating Flip-Flop logic design.
   Simulating Ripple Carry 8-bit adder logic design.
b) Flip-Flop
   Ripple Carry
c) Error
d) None of the above

143. What will be the output of the following code snippet?

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

    def calculate_delay(self, gate_count):
        delay = 0.5 * gate_count
        return f"Calculating delay for {self.name}: {delay} ns"

    def calculate_power(self, gate_count, voltage):
        power = gate_count * voltage
        return f"Calculating power consumption for {self.name}: {power} W"
```

```
calculator = LogicCalculator("AND Gate")

delay_result = calculator.calculate_delay(10)
power_result = calculator.calculate_power(10, 1.8)

print(delay_result)
print(power_result)
```

a) Calculating delay for AND Gate: 5.0 ns
   Calculating power consumption for AND Gate: 18.0 W
b) AND Gate
   AND Gate
c) Error
d) None of the above

144. What is the output of the following code:

```
class MyList:
    def __init__(self, lst):
        self.lst = lst

    def __getitem__(self, index):
        return self.lst[index]

a = MyList([1, 2, 3, 4])
print(a[1:3])
```

a) [2, 3]
b) [1, 2, 3]
c) [2, 3, 4]
d) TypeError

145. What is the output of the following code:

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

    def __repr__(self):
        return self.value.upper()

a = MyString("hello")
print(repr(a))
```

a) "hello"
b) "HELLO"
c) <__main__.MyString object at 0x7f2d2a9f9e10>
d) TypeError

146. What is the output of the following Python code?

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

    def __eq__(self, other):
        if isinstance(other, MyString):
            return self.value.lower() == other.value.lower()
        elif isinstance(other, str):
            return self.value.lower() == other.lower()
        else:
            return False

a = MyString("Hello")
b = MyString("hello")

if a == "HELLO":
    print("Match")
else:
    print("No match")
```

a) Match
b) No match
c) TypeError
d) AttributeError

147. What will be the output of the following code:

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

    def move(self):
        print(f"{self.name} is moving.")

class TalkingRobot(Robot):
    def __init__(self, name, language):
        super().__init__(name)
        self.language = language

    def talk(self):
        print(f"{self.name} is talking in {self.language}.")

class DancingRobot(Robot):
    def dance(self):
        print(f"{self.name} is dancing.")

class SuperRobot(TalkingRobot, DancingRobot):
    def __init__(self, name, language, power):
        TalkingRobot.__init__(self, name, language)
        self.power = power

    def superpower(self):
        print(f"{self.name} has superpower of {self.power}.")

robot1 = SuperRobot("Tom", "English", "flying")
robot2 = DancingRobot("Jerry")

robots = [robot1, robot2]

for robot in robots:
    robot.move()
```

a) Tom is talking in English. Jerry is dancing.
b) Tom is moving. Jerry is dancing.
c) Tom is moving. Jerry is moving.
d) Tom is dancing. Jerry is moving.

148. **What will be the output of the following code:**

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

    def __str__(self):
        return f"I am a robot named {self.name}."

    def __eq__(self, other):
        return self.name == other.name

    def __add__(self, other):
        return Robot(f"{self.name}-{other.name}")

robot1 = Robot("Robo")
robot2 = Robot("Bot")
robot3 = Robot("Bot")

print(robot1 == robot2)
print(robot2 == robot3)

combined_robot = robot1 + robot2
print(combined_robot)
```

a) **False, False, I am a robot named Robo-Bot.**
b) **False, True, I am a robot named Robo-Bot.**
c) **True, False, I am a robot named Robo-Bot.**
d) **True, True, I am a robot named Robo-Bot.**

149. **What will be the output of the following code:**

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

    def display(self):
        print("A:", self.x)

class B(A):
    def __init__(self):
        super().__init__()
        self.x = 20

    def display(self):
        print("B:", self.x)

obj = B()
obj.display()
```

a) A: 10
b) B: 10
c) A: 20
d) B: 20

150. What will be the output of the following code:

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

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

obj = B()
obj.print()
```

a) A
b) B
c) B A
d) A B

151. What will be the output of the following code:

```
class A:
    @staticmethod
    def display():
        print("A")

class B(A):
    def display():
        print("B")

obj = B()
obj.display()
```

a) A
b) B
c) Time Limit
d) TypeError

152. **What will be the output of the following code:**

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

    def display(self):
        print("A:", self.x)

class B(A):
    def __init__(self):
        super().__init__()
        self.x = 20

obj1 = A()
obj2 = B()
obj1.display()
obj2.display()
```

a) **A: 10, B: 10**
b) **B:10, b: 20**
c) **A: 10, A: 20**
d) **B: 20, B: 20**

153. **What will be the output of the following code:**

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

    def display(self):
        print("A:", self.x)

class B(A):
    def __init__(self):
        self.x = 20
        super().__init__()

obj = B()
obj.display()
```

a) **A: 10**
b) **A: 20**
c) **B: 20**
d) **B: 10**

154. **What will be the output of the following code:**

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

    def display(self):
        print("A:", self.x)

class B(A):
    def __init__(self):
        super().__init__()
        self.x = 20

    def display(self):
        print("B:", self.x)

obj = B()
obj.display()
```

a) B: 10
b) B: 20
c) Compilation error
d) Runtime exception

155. **What will be the output of the following code:**

```
class A:
    def __init__(self, x):
        self.x = 5

    def display(self):
        print(self.x)

class B(A):
    def __init__(self, x, y):
        self.y = y
        super().__init__(x)

    def display(self):
        print(self.y)
        super().display()

obj = B(10, 20)
obj.display()
```

a) 20, 10
b) 10, 20
c) 20, 5
d) Compilation error

156. What will be the output of the following code:

```
class Room:
    def __init__(self, number):
        self.number = number

    def display(self):
        print("Room number:", self.number)

class Hotel:
    def __init__(self):
        self.rooms = []

    def add_room(self, room):
        self.rooms.append(room)

    def display_rooms(self):
        for room in self.rooms:
            room.display()


hotel = Hotel()
room1 = Room(101)
room2 = Room(102)
hotel.add_room(room1)
hotel.add_room(room2)
hotel.display_rooms()
```

a) Room number: 101, Room number: 102
b) 101, 102
c) Compilation error
d) Runtime exception

157. What will be the output of the following code:

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

    def display(self):
        print("A:", self.x)

class B(A):
    def __init__(self):
        super().__init__()
        self.x = 20

    def display(self):
        super().display()
        print("B:", self.x)

obj = B()
obj.display()
```

a) A: 20, B: 10
b) A: 20, B: 20
c) A: 10, B: 20
d) Compilation error

158. What will be the output of the following code:

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

    def __str__(self):
        return self.name


class Product:
    def __init__(self, name, factory):
        self.name = name
        self.factory = factory

    def display(self):
        print("Product:", self.name)
        print("Factory:", self.factory)
```

```
f1 = Factory("Factory A")
f2 = Factory("Factory B")
p1 = Product("Product 1", f2)
p2 = Product("Product 2", f1)
p1.display()
p2.display()
```

a) Product: Product 1, Factory: Factory A, Product: Product 2, Factory: Factory B
b) Product: Product 1, Factory: Factory B, Product: Product 2, Factory: Factory A
c) Compilation error
d) Runtime exception

159. What will be the output of the following code:

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

    def display(self):
        print("Company:", self.name)


class Employee:
    def __init__(self, name, company):
        self.name = name
        self.company = company

    def display(self):
        print("Employee:", self.name)
        self.company.display()


company = Company("XYZ Corp")
employee1 = Employee("John", company)
employee2 = Employee("Tom", company)
employee3 = Employee("Naij", company)
employee.display()
```

a) Employee: John, Company: XYZ Corp
b) Employee: Tom, Company: XYZ Corp
c) Employee: Naij, Company: XYZ Corp
d) NameError: name 'employee' is not defined

160. What will be the output of the following code:

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

    def __repr__(self):
        return self.name


class Product:
    def __init__(self, name, shop):
        self.name = name
        self.shop = shop

    def display(self):
        print("Product:", self.name)
        print("Shop:", self.shop)


s1 = Shop("Shop A")
s2 = Shop("Shop B")
p1 = Product("Product 1", s1)
p2 = Product("Product 2", s2)
p1.display()
p2.display()
```

a) Product: Product 1, Shop: Shop A, Product: Product 2, Shop: Shop B
b) Product 1, Shop A, Product 2, Shop B
c) Compilation error
d) Runtime exception

161. What will be the output of the following code:

```
class Laptop:
    def __init__(self, brand):
        self.brand = brand

    def display(self):
        print("Brand:", type(self).__name__)


class MacBook(Laptop):
    def __init__(self, brand, year):
        super().__init__(brand)
        self.year = year + 1

    def display(self):
        super().display()
        print("Year:", self.year)


macbook = MacBook("Apple", 2022)
macbook.display()
```

a) **Brand: Apple, Year: 2022**
b) **Brand: MacBook, Year: 2022**
c) **Brand MacBooK, Year: 2023**
d) **Brand: Apple, Year: 2023**

162. What will be the output of the following code:

```
class Faculty:
    def __init__(self, name):
        self.name = name
        self.courses = []

    def add_course(self, course):
        self.courses.append(course)

    def display(self):
        print("Faculty Name:", self.name)
        print("Courses:", ", ".join(self.courses))


class Professor(Faculty):
    def __init__(self, name, department):
        super().__init__(name)
        self.department = department

    def display(self):
        super().display()
        print("Department:", self.department)


faculty = Faculty("FAI")
faculty.add_course("Linear Algebra")
faculty.add_course("Statistics ")
professor = Professor("Mohamed Fathy", "Computer Science")
professor.add_course("Programming")
faculty.display()
professor.display()
```

a) **Faculty Name: Mohamed Fathy, Courses: Programming, Faculty Name: FAI, Courses: Linear Algebra, Statistics, Department: Computer Science**
b) **Faculty Name: FAI, Courses: : Linear Algebra, Statistics, Faculty Name: Mohamed Fathy, Courses: Programming, , Department: Computer Science**
c) **Compilation error**
d) **Runtime exception**

163. What will be the output of the following code:

```
class Race:
    def __init__(self, distance):
        self.distance = distance
        self.participants = []

    def add_participant(self, participant):
        self.participants.append(participant)

    def display(self):
        print("Race Distance:", self.distance)
        print("Participants:", ", ".join(str(p) for p in self.participants))


class Runner:
    def __init__(self, name, speed):
        self.name = name
        self.speed = speed

    def __str__(self):
        return self.name

    def time_to_finish(self, distance):
        return round(distance/self.speed, 2)


race = Race(1000)
runner1 = Runner("John", 10)
runner2 = Runner("Jane", 15)
race.add_participant(runner1)
race.add_participant(runner2)
race.display()
print("Time to finish (John):", runner1.time_to_finish(1000))
print("Time to finish (Jane):", runner2.time_to_finish(1000))
```

a) **Race Distance: 1000, Participants: John, Jane, Time to finish (John): 100.0, Time to finish (Jane): 150.0**
b) **Race Distance: 1000, Participants: John, Jane, Time to finish (John): 100.0, Time to finish (Jane): 66.67**
c) **Compilation error**
d) **Runtime exception**

164. What will be the output of the following code:

```
class Islam:
    def __init__(self):
        self.name = "Islam"
        self.pillars = []

    def add_pillar(self, pillar):
        self.pillars.append(pillar)

    def display(self):
        print("Religion:", self.name)
        print("Pillars:", ", ".join(self.pillars))


class Muslim(Islam):
    def __init__(self, name):
        super().__init__()
        self.name = name


islam = Islam()
islam.add_pillar("Shahada")
islam.add_pillar("Salah")

muslim = Muslim("Ahmed")
muslim.add_pillar("Zakat")
muslim.add_pillar("Sawm")

islam.display()
muslim.display()
```

a) **Religion: Islam, Pillars: Zakat, Sawm, Religion: Islam, Pillars: Shahada, Salah**
b) **Religion: Ahmed, Pillars: Shahada, Salah, Religion: Islam, Pillars: Zakat, Sawm**
c) **Religion: Islam, Pillars: Shahada, Salah, Religion: Ahmed, Pillars: Zakat, Sawm**
d) **Religion: Ahmed, Pillars: Zakat, Sawm, Religion: Islam, Pillars: Shahada, Salah**

165. What will be the output of the following code:

```
class DataScience:
    def __init__(self, tools):
        self.tools = tools

    def __str__(self):
        return "Data Science: " + ", ".join(self.tools)


class MachineLearning(DataScience):
    def __init__(self, tools, algorithms):
        super().__init__(tools)
        self.algorithms = algorithms

    def __str__(self):
        return super().__str__() + ", Machine Learning: " + f'{["
".join(self.algorithms)]}'


class DeepLearning(MachineLearning):
    def __init__(self, tools, algorithms, layers):
        super().__init__(tools, algorithms)
        self.layers = layers

    def __str__(self):
        return super().__str__() + ", Deep Learning Layers: " +
str(self.layers)


deep_learning = DeepLearning(["Python", "TensorFlow"], ["Convolutional Neural
Networks"], 5)
print(deep_learning)
```

a) **Data Science: Python, TensorFlow, Machine Learning: Convolutional Neural Networks, Deep Learning Layers: 5**
b) **Data Science: Python, TensorFlow, Machine Learning: Convolutional Neural Networks, Deep Learning Layers: [5]**
c) **Data Science: Python, TensorFlow, Machine Learning: ['Convolutional Neural Networks'], Deep Learning Layers: 5**
d) **Compilation error**

166. What will be the output of the following code:

```
class DataScience:
    def __init__(self, tools):
        self.tools = tools

    def __str__(self):
        return "Data Science Tools: " + ", ".join(self.tools)


class MachineLearning(DataScience):
    def __init__(self, tools, algorithms):
        super().__init__(tools)
        self.algorithms = algorithms


class DeepLearning(MachineLearning):
    def __init__(self, tools, algorithms, layers):
        super().__init__(tools, algorithms)
        self.layers = layers


deep_learning = DeepLearning(["Python", "TensorFlow"], ["Convolutional Neural Networks"], 5)
print(deep_learning)
```

a) **Data Science: Python, TensorFlow, Machine Learning: Convolutional Neural Networks, Deep Learning Layers: 5**
b) **Data Science Tools: Python, TensorFlow**
c) **Data Science: Python, TensorFlow, Machine Learning: ['Convolutional Neural Networks'], Deep Learning Layers: 5**
d) **Compilation error**

167. **What will be the output of the following code:**

```
class CyberSecurity:
    def __init__(self, tools):
        self.tools = tools

    def __str__(self):
        return "Cybersecurity Tools: " + ", ".join(self.tools)


class Encryption(CyberSecurity):
    def __init__(self, tools, algorithm):
        super().__init__(tools)
        self.algorithm = algorithm

    def __str__(self):
        return super().__str__() + ", Encryption Algorithm: " + self.algorithm


class Firewall(CyberSecurity):
    def __init__(self, tools, port):
        super().__init__(tools)
        self.port = port

    def __str__(self):
        return super().__str__() + ", Firewall Port: " + str(self.port)


cybersec = CyberSecurity(["Nmap", "Wireshark"])
encryption = Encryption(["Nmap", "Wireshark"], "AES")
firewall = Firewall(["Nmap", "Wireshark"], 8080)

print(cybersec)
print(encryption)
print(firewall)
```

a) **Cybersecurity Tools: Nmap, Wireshark, Encryption Algorithm: AES**
   Cybersecurity Tools: Nmap, Wireshark
   Cybersecurity Tools: Nmap, Wireshark, Firewall Port: 8080

b) **Cybersecurity Tools: Nmap, Wireshark**
   Cybersecurity Tools: Nmap, Wireshark, Encryption Algorithm: AES
   Cybersecurity Tools: Nmap, Wireshark, Firewall Port: 8080

c) **Cybersecurity Tools: Nmap, Wireshark, Firewall Port: 8080**
   Cybersecurity Tools: Nmap, Wireshark, Encryption Algorithm: AES
   Cybersecurity Tools: Nmap, Wireshark

d) **Compilation error**

168. **What will be the output of the following code:**

```
class CyberSecurity:
    def __init__(self, tools):
        self.tools = tools

    def __str__(self):
        return "Cybersecurity Tools: " + ", ".join(self.tools)


class Encryption(CyberSecurity):
    def __init__(self, tools, algorithm):
        super().__init__(tools)
        self.algorithm = algorithm

    def __str__(self):
        return super().__str__() + ", Encryption Algorithm: " + self.algorithm


class Firewall(CyberSecurity):
    def __init__(self, tools, port):
        super().__init__(tools)
        self.port = port

    def __str__(self):
        return super().__str__() + ", Firewall Port: " + str(self.port)


cybersec = CyberSecurity(["Nmap", "Wireshark"])
encryption = Encryption(["Nmap", "Wireshark"], "AES")
firewall = Firewall(["Nmap", "Wireshark"], 8080)

print(list(cybersec))
print(encryption)
print(firewall)
```

a) **Cybersecurity Tools: Nmap, Wireshark**
Cybersecurity Tools: Nmap, Wireshark, Encryption Algorithm: AES
Cybersecurity Tools: Nmap, Wireshark, Firewall Port: 8080

b) **Cybersecurity Tools: Nmap, Wireshark, Encryption Algorithm: AES**
Cybersecurity Tools: Nmap, Wireshark, Firewall Port: 8080
Cybersecurity Tools: Nmap, Wireshark

c) **Cybersecurity Tools: Nmap, Wireshark**
Cybersecurity Tools: Nmap, Wireshark, Firewall Port: 8080
Cybersecurity Tools: Nmap, Wireshark, Encryption Algorithm: AES

d) **TypeError: 'CyberSecurity' object is not iterable**

169. What will be the output of the following code:

```
class Encryption:
    def __init__(self, key):
        self.key = key

    def encrypt(self, message):
        encrypted_message = ""
        for char in message:
            encrypted_char = chr(ord(char) + self.key)
            encrypted_message += encrypted_char
        return encrypted_message


class AdvancedEncryption(Encryption):
    def encrypt(self, message):
        encrypted_message = super().encrypt(message)
        reversed_message = encrypted_message[::-1]
        return reversed_message


encryption = Encryption(3)
advanced_encryption = AdvancedEncryption(5)

message = "Hello, World!"
encrypted_message = encryption.encrypt(message)
advanced_encrypted_message = advanced_encryption.encrypt(message)

print(encrypted_message)
print(advanced_encrypted_message)
```

a) Khoor/#Brxog:
   !dlroW ,olleH
b) Khoor/#Sdht$
   &iqwt\%1tqqjM
c) Khoor/#Zruog$
   &iqwt\%1tqqjM
d) Compilation error

170. What will be the output of the following code:

```
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None


class PrimeNumberTree:
    def __init__(self):
        self.root = None

    def insert(self, value):
        if self.root is None:
            self.root = TreeNode(value)
        else:
            self._insert_recursive(self.root, value)

    def _insert_recursive(self, node, value):
        if value < node.value:
            if node.left is None:
                node.left = TreeNode(value)
            else:
                self._insert_recursive(node.left, value)
        else:
            if node.right is None:
                node.right = TreeNode(value)
            else:
                self._insert_recursive(node.right, value)

    def inorder_traversal(self):
        if self.root is None:
            return []
        else:
            return self._inorder_recursive(self.root)

    def _inorder_recursive(self, node):
        if node is None:
            return []
        else:
            left_subtree = self._inorder_recursive(node.left)
            right_subtree = self._inorder_recursive(node.right)
            return left_subtree + [node.value] + right_subtree


prime_tree = PrimeNumberTree()
```

```
prime_tree.insert(5)
prime_tree.insert(7)
prime_tree.insert(2)
prime_tree.insert(3)
prime_tree.insert(11)

traversal_result = prime_tree.inorder_traversal()
print(traversal_result)
```

```
# The shape of tree
            5
           / \
          2   7
           \   \
            3   11
```

a) **[2, 5, 7, 3, 11]**
b) **[11, 7, 5, 3, 2]**
c) **[2, 3, 5, 7, 11]**
d) **[3, 2, 5, 7, 11]**

171. What will be the output of the following code:

```
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None


class BinaryTree:
    def __init__(self, root):
        self.root = TreeNode(root)

    def preorder_traversal(self):
        return self._preorder_recursive(self.root)

    def _preorder_recursive(self, node):
        if node is None:
            return []
        else:
            left_subtree = self._preorder_recursive(node.left)
            right_subtree = self._preorder_recursive(node.right)
            return [node.value] + left_subtree + right_subtree
```

```
binary_tree = BinaryTree(1)
binary_tree.root.left = TreeNode(2)
binary_tree.root.right = TreeNode(3)
binary_tree.root.left.left = TreeNode(4)
binary_tree.root.left.right = TreeNode(5)

traversal_result = binary_tree.preorder_traversal()
print(traversal_result)
```

```
# The shape of tree
        1
       / \
      2   3
     / \
    4   5
```

a) [1, 2, 4, 5, 3]
b) [4, 2, 5, 1, 3]
c) [1, 3, 2, 5, 4]
d) [1, 2, 3, 4, 5]

172. **What will be the output of the following code:**

```
import numpy as np

class NeuralNetwork:
    def __init__(self):
        self.weights = np.array([0.5, -0.5])
        self.bias = 0.1

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def forward(self, input):
        weighted_sum = np.dot(input, self.weights) + self.bias
        output = self.sigmoid(weighted_sum)
        return output


neural_net = NeuralNetwork()
input_data = np.array([0.2, 0.4])

output = neural_net.forward(input_data)

print(output)
```

a) 0.475
b) 0.50
c) 0.55
d) 0.60

173. What will be the output of the following code if the initial value is 42:

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

    def to_binary(self):
        return bin(self.value)

    def to_octal(self):
        return oct(self.value)

    def to_hexadecimal(self):
        return hex(self.value)


number = 42
converter = NumberSystemConverter(number)

binary = converter.to_binary()
octal = converter.to_octal()
hexadecimal = converter.to_hexadecimal()

print(binary)
print(octal)
print(hexadecimal)
```

a) 0b101010, 0o42, 0x2a
b) 0b101010, 0o52, 0x2a
c) 0b101010, 0o52, 0x42
d) 0b101010, 0o42, 0x42

174. What will be the output of the following code:

```
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def get_brand(self):
        return self.brand

    def get_model(self):
        return self.model


class SportsCar(Car):
    def __init__(self, brand, model, top_speed):
        super().__init__(brand, model)
        self.top_speed = top_speed

    def get_top_speed(self):
        return self.top_speed


car1 = Car("Toyota", "Corolla")
car2 = SportsCar("Ferrari", "488 GTB", 330)

print(car1.get_brand())
print(car2.get_model())
print(car2.get_top_speed())
```

a) Toyota, 488 GTB, 330
b) Corolla, Ferrar, 488 GTB, 330
c) Toyota, Ferrari, 330
d) Corolla, 488 GTB, 330

175. What will be the output of the following code if the initial matrix is [[1, 2], [3, 4]]:

```
import numpy as np

class LinearAlgebra:
    def __init__(self, matrix):
        self.matrix = matrix

    def matrix_transpose(self):
        return np.transpose(self.matrix)

    def matrix_inverse(self):
        return np.linalg.inv(self.matrix)

    def matrix_determinant(self):
        return abs(round(np.linalg.det(self.matrix), 2))

matrix = np.array([[1, 2], [3, 4]])
linear_algebra = LinearAlgebra(matrix)

transpose = linear_algebra.matrix_transpose()
inverse = linear_algebra.matrix_inverse()
determinant = linear_algebra.matrix_determinant()

print(transpose)
print(inverse)
print(determinant)
```

a) **[[1, 2], [2, 4]], [[-2, 1], [1.5, -0.5]], 2.0**
b) **[[1, 3], [2, 4]], [[-2, 1], [1.5, -0.5]], 2.0**
c) **[[4, 3], [2, 1]], [[-2, 1], [1.5, -0.5]], 2.0**
d) **[[4, 2], [3, 1]], [[-2, 1], [1.5, -0.5]], 2.0**

176. What will be the output of the following code if the initial data is [10, 15, 20, 25, 30]:

```
import statistics

class StatisticsCalculator:
    def __init__(self, data):
        self.data = data

    def calculate_mean(self):
        return statistics.mean(self.data)

    def calculate_median(self):
        return statistics.median(self.data)

    def calculate_standard_deviation(self):
        return round(statistics.stdev(self.data), 2)

data = [10, 15, 20, 25, 30]
stats_calc = StatisticsCalculator(data)

mean = stats_calc.calculate_mean()
median = stats_calc.calculate_median()
std_deviation = stats_calc.calculate_standard_deviation()

print(mean)
print(median)
print(std_deviation)
```

a) **20, 22.5, 7.91**
b) **20, 22.5, 9.03**
c) **20, 20, 7.91**
d) **22.5, 20, 7.91**

177. What will be the output of the following code if the initial data is [10, 15, 20, 25, 30]:

```
class Statistics:
    def __init__(self, data):
        self.data = data

    def calculate(self):
        return "Calculating statistics"


class MeanCalculator(Statistics):
    def calculate(self):
        return sum(self.data) / len(self.data)


class MedianCalculator(Statistics):
    def calculate(self):
        sorted_data = sorted(self.data)
        mid_index = len(sorted_data) // 2
        if len(sorted_data) % 2 == 0:
            return (sorted_data[mid_index - 1] + sorted_data[mid_index]) / 2
        else:
            return sorted_data[mid_index]


data = [10, 15, 20, 25, 30]
mean_calc = MeanCalculator(data)
median_calc = MedianCalculator(data)

mean = mean_calc.calculate()
median = median_calc.calculate()

print(mean)
print(median)
```

a) 20, 20.0
b) 20, 22.5
c) 20.0, 25
d) 20.0, 20

178. What will be the output of the following code:

```
class Statistics:
    def __init__(self, data):
        self.data = data

    def calculate(self):
        return "Calculating statistics"

class Probability(Statistics):
    def calculate(self):
        return "Calculating probability"

class Distribution(Statistics):
    def calculate(self):
        return "Calculating distribution"

class Estimation(Statistics):
    def calculate(self):
        return "Calculating estimation"

class HypothesisTesting(Statistics):
    def calculate(self):
        return "Performing hypothesis testing"

data = [10, 15, 20, 25, 30]
probability_calc = Probability(data)
distribution_calc = Distribution(data)
estimation_calc = Estimation(data)
hypothesis_testing_calc = HypothesisTesting(data)

probability_result = probability_calc.calculate()
distribution_result = distribution_calc.calculate()
estimation_result = estimation_calc.calculate()
hypothesis_testing_result = hypothesis_testing_calc.calculate()

print(probability_result)
print(distribution_result)
print(estimation_result)
print(hypothesis_testing_result)
```

a) Calculating probability, Calculating distribution, Calculating estimation, Performing hypothesis testing
b) Calculating statistics, Calculating statistics, Calculating statistics, Calculating statistics
c) Calculating probability, Calculating distribution, Calculating estimation, Calculating statistics
d) Calculating statistics, Calculating statistics, Calculating statistics, Performing hypothesis testing

179. What will be the output of the following code if the initial data is [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:

```
import statistics
from scipy.stats import ttest_1samp

class Statistics:
    def __init__(self, data):
        self.data = data

    def calculate(self):
        return "Calculating statistics"

class Probability(Statistics):
    def calculate(self):
        return len(self.data) / 10

class Distribution(Statistics):
    def calculate(self):
        return statistics.mean(self.data)

class Estimation(Statistics):
    def calculate(self):
        return statistics.stdev(self.data)

class HypothesisTesting(Statistics):
    def calculate(self):
        return ttest_1samp(self.data, 0)

data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
probability_calc = Probability(data)
distribution_calc = Distribution(data)
estimation_calc = Estimation(data)
hypothesis_testing_calc = HypothesisTesting(data)

probability_result = probability_calc.calculate()
distribution_result = distribution_calc.calculate()
estimation_result = estimation_calc.calculate()
hypothesis_testing_result = hypothesis_testing_calc.calculate()

print(probability_result)
print(distribution_result)
print(estimation_result)
print(hypothesis_testing_result)
```

a) 1.0, 5.5, 3.0276503540974917, TtestResult(statistic=7.544562646538029, pvalue=0.00027819601104818546, df=9)
b) 1.0, 5.5, 3.0276503540974917, TtestResult(statistic=5.744562646538029, pvalue=0.00027819601104818546, df=9)
c) 10, 5.5, 3.0276503540974917, TtestResult(statistic=5.744562646538029, pvalue=0.00027819601104818546, df=9)
d) AttributeError: module 'statistics' has no attribute 'ttest_1samp'

180. What will be the output of the following code:

```
class Magnet:
    def __init__(self, strength):
        self.strength = strength

    def attract(self, other_magnet):
        return self.strength * other_magnet.strength

    def __str__(self):
        return f"Magnet with strength {self.strength}"

class Electromagnet(Magnet):
    def __init__(self, strength, current):
        super().__init__(strength)
        self.current = current

    def attract(self, other_magnet):
        return self.strength * other_magnet.strength * self.current

class PermanentMagnet(Magnet):
    def __init__(self, strength, orientation):
        super().__init__(strength)
        self.orientation = orientation

    def new_strength(self, new):
        self.strength = new

magnet1 = Magnet(5)
magnet2 = Electromagnet(3, 10)
magnet3 = PermanentMagnet(8, True)

attract1 = magnet1.attract(magnet2)
attract2 = magnet2.attract(magnet3)
attract3 = magnet3.attract(magnet1)

magnet3.new_strength(7)
attract4 = magnet3.attract(magnet1)
```

```
print(attract1)
print(attract2)
print(attract3)
print(attract4)
```

a) 15, 240, 40, 40
b) 15, 240, 40, 35
c) 50, 240, 40, 40
d) 50, 240, 40, 35

181. **What will be the output of the following code:**

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

    def process_data(self):
        return "Data processed by AI"

    def __str__(self):
        return f"I am {self.name}, an AI"

class MachineLearning(AI):
    def process_data(self):
        return "Data processed by Machine Learning"

class DeepLearning(AI):
    def process_data(self):
        return "Data processed by Deep Learning"

ai1 = AI("AI 1")
ml = MachineLearning("ML 1")
dl = DeepLearning("DL 1")

print(ai1)
print(ml.process_data())
print(dl.process_data())
```

a) Data processed by AI, Data processed by Machine Learning, Data processed by Deep Learning
b) Data processed by AI, Data processed by AI, Data processed by AI
c) I am AI 1, an AI, Data processed by Machine Learning, Data processed by Deep Learning
d) I am AI 1, an AI, Data processed by AI, Data processed by AI

182. What will be the output of the following code:

```
class Quran:
    def __init__(self, verse_arabic, verse_english, information):
        self.verse_arabic = verse_arabic
        self.verse_english = verse_english
        self.information = information


class AyatAlKursi(Quran):
    def __init__(self, verse_arabic, verse_english, information):
        super().__init__(verse_arabic, verse_english, information)

    def display_verse(self):
        print("Ayat al-Kursi (Verse of the Chair):\n")
        print("Arabic: ", self.verse_arabic)
        print("English: ", self.verse_english)
        print("Information: ", self.information)


verse_arabic = " اللَّهُ لَا إِلَٰهَ إِلَّا هُوَ الْحَيُّ الْقَيُّومُ ۚ لَا تَأْخُذُهُ سِنَةٌ وَلَا نَوْمٌ ۚ لَّهُ مَا فِي السَّمَاوَاتِ وَمَا فِي الْأَرْضِ ۗ مَن ذَا الَّذِي يَشْفَعُ
عِندَهُ إِلَّا بِإِذْنِهِ ۚ يَعْلَمُ مَا بَيْنَ أَيْدِيهِمْ وَمَا خَلْفَهُمْ ۖ وَلَا يُحِيطُونَ بِشَيْءٍ مِّنْ عِلْمِهِ إِلَّا بِمَا شَاءَ ۚ وَسِعَ كُرْسِيُّهُ السَّمَاوَاتِ وَالْأَرْضَ ۖ وَلَا يَئُودُهُ
حِفْظُهُمَا ۚ وَهُوَ الْعَلِيُّ الْعَظِيمُ"

verse_english = "Allah! There is no deity except Him, the Ever-Living, the
Sustainer of existence. Neither drowsiness overtakes Him nor sleep. To Him
belongs whatever is in the heavens and whatever is on the earth. Who is it that
can intercede with Him except by His permission? He knows what is before them
and what will be after them, and they encompass not a thing of His knowledge
except for what He wills. His Kursi extends over the heavens and the earth, and
their preservation tires Him not. And He is the Most High, the Most Great."

information = "Ayat al-Kursi is verse 255 of Surah Al-Baqarah in the Quran. It
is one of the most significant verses in Islamic belief, highlighting the
absolute sovereignty, knowledge, and power of Allah. It describes His
attributes, including His eternal existence, omniscience, and control over the
heavens and the earth. Ayat al-Kursi is often recited for protection and is
considered a means of seeking blessings and divine guidance."

ayat = AyatAlKursi(verse_arabic, verse_english, information)
ayat.display_verse()
```

a) It will show Ayat AlKursi in Arabic and English and its information.
b) Error: The code will throw an error due to incorrect inheritance usage.
c) Ayat al-Kursi
d) dNone of the above

Exceptions:

183. What is an exception in Python?

a) A syntax error in Python code.
b) A logical error in Python code.
c) An event that interrupts the normal flow of a Python program.
d) A keyword in Python used for error handling.

184. Which of the following is not a built-in exception in Python?

a) ZeroDivisionError
b) IndexError
c) KeyNotFoundException
d) TypeError

185. What is the output of the following code?

```
try:
    print("Hello" + 5)
except TypeError:
    print("Type Error occurred")
```

a) Hello5
b) 5Hello
c) ValueError
d) Type Error occurred

186. Which of the following is not a type of exception in Python?

a) SyntaxError
b) ValueError
c) TypeError
d) BooleanError

187. What is the output of the following code?

```
try:
    a = 5/0
except ZeroDivisionError:
    print("Zero Division Error occurred")
finally:
    print("Finally block executed")
```

a) Zero Division Error occurred
b) Finally block executed
c) Zero Division Error occurred, Finally block executed
d) None of the above

188. Which of the following statements about custom exceptions in Python is true?
a) Custom exceptions cannot be raised in Python.
b) Custom exceptions must always inherit from the built-in Exception class.
c) Custom exceptions must always inherit from the built-in BaseException class.
d) Custom exceptions can only be raised in object-oriented Python code.

189. Which of the following is an example of an object-oriented approach to error handling in Python?
a) Using try-except blocks in procedural code.
b) Raising built-in exceptions in procedural code.
c) Creating custom exceptions and using them in object-oriented code.
d) All of the above.

190. What is the output of the following code?

```
class MyException(Exception):
    pass

try:
    raise MyException("This is a custom exception")
except MyException as e:
    print(e)
```

a) This is a custom exception
b) MyException
c) NameError: MyException is not defined
d) None of the above

191. What is the output of the following code?

```
class MyException(Exception):
    def __init__(self, message):
        self.message = message

try:
    raise MyException("This is a custom exception")
except MyException as e:
    print(e.message)
```

a) This is a custom exception
b) MyException
c) NameError: MyException is not defined
d) None of the above

192. What is the output of the following code:

```
class MyException(Exception):
    def __init__(self, message):
        self.message = message

class AnotherException(Exception):
    pass

try:
    raise AnotherException("This is another exception")
except MyException as e:
    print("MyException occurred")
except Exception as e:
    print("Exception occurred")
```

a) MyException occurred
b) Exception occurred
c) This code will raise an error because AnotherException is not caught.
d) None of the above

193. What is the output of the following code:

```
class MyException(Exception):
    pass

class AnotherException(MyException):
    pass

try:
    raise AnotherException("This is another exception")
except MyException as e:
    print("MyException occurred")
except Exception as e:
    print("Exception occurred")
```

a) MyException occurred
b) Exception occurred
c) This code will raise an error because AnotherException is not caught.
d) None of the above

194. What is the output of the following code?

```
class MyError:
    def __init__(self, message):
        self.message = message

try:
    raise MyError("This is a custom error")
except MyError as e:
    print(e.message)
```

a) This is a custom error
b) MyError
c) AttributeError: 'Exception' object has no attribute 'message'
d) TypeError: catching classes that do not inherit from BaseException is not allowed

195. What is the output of the following code?

```
class MyError:
    def __init__(self, message):
        self.message = message

try:
    raise MyError("This is a custom error")
except Exception as e:
    print(e.message)
```

a) MyError occurred
b) AttributeError: 'TypeError' object has no attribute 'message'
c) This code will raise an error because AnotherError is not caught.
d) None of the above

196. What is the output of the following code?

```
class MyError:
    def __init__(self, message):
        self.message = message

class AnotherError:
    pass

try:
    raise AnotherError("This is another error")
except MyError as e:
    print("MyError occurred")
except Exception as e:
    print("Exception occurred")
```

a) MyError occurred
b) Exception occurred
c) This code will raise an error because AnotherError is not caught
d) None of the above

197. What is the output of the following code?

```
class CarError:
    pass

class EngineError(CarError):
    pass

class TransmissionError(CarError):
    pass

try:
    raise EngineError("The engine has failed")
except TransmissionError as e:
    print("TransmissionError occurred")
except CarError as e:
    print("CarError occurred")
```

a) TransmissionError occurred
b) CarError occurred
c) This code will raise an error because EngineError is not caught
d) None of the above

198. What is the output of the following code?

```
class Person:
    def __init__(self, name, age):
        if not isinstance(name, str):
            raise TypeError("Name must be a string")
        if not isinstance(age, int):
            raise TypeError("Age must be an integer")
        self.name = name
        self.age = age

try:
    p = Person("Alice", "100")
except TypeError as e:
    print(e)
```

a) Name must be a string
b) Age must be an integer
c) This code will raise an error because PersonError is not defined.
d) None of the above

199. What is the output of the following code?

```
class Account:
    def __init__(self, balance):
        if balance < 0:
            raise TypeError("Balance cannot be negative")
        self.balance = balance

try:
    a = Account(-100)
except TypeError as e:
    print(e)
except Exception as e:
    print("Exception occurred")
```

a) Balance cannot be negative
b) BankError occurred
c) Exception occurred
d) None of the above

200. What is the output of the following code?

```
class BankIssue:
    def __init__(self, message):
        self.message = message

class Account:
    def __init__(self, balance):
        if balance < 0:
            raise BankIssue("Balance cannot be negative")
        self.balance = balance

try:
    a = Account(-100)
except BankIssue as e:
    print(e.message)
except Exception as e:
    print("Exception occurred")
```

a) Balance cannot be negative
b) BankIssue occurred
c) Exception occurred
d) TypeError: catching classes that do not inherit from BaseException is not allowed

201. What is the output of the following code?

```
try:
    num = int('abc')
except ValueError:
    print("Error: Invalid literal for int()")
```

a) ValueError
b) Error: Invalid literal for int()
c) Error: unsupported operand type(s) for /: 'int' and 'str'
d) None of the above

202. What is the output of the following code?

```
try:
    lst = [1, 2, 3]
    print(lst[-3])
except IndexError:
    print("Error: List index out of range")
```

a) Error: List index out of range
b) IndexError
c) 1
d) None of the above

203. What is the output of the following code?

```
try:
    file = open("nonexistent.txt", "r")
except FileNotFoundError:
    print("Error: File not found")
```

a) Error: File not found
b) FileNotFoundError
c) Error: 'module' object has no attribute 'open'
d) None of the above

204. What is the output of the following code?

```
try:
    x = 5 + "abc"
except ValueError:
    print("Error: unsupported operand type(s) for +: 'int' and 'str'")
```

a) Error: unsupported operand type(s) for +: 'int' and 'str'
b) TypeError
c) Error: 'int' object is not iterable
d) None of the above

205. What is the output of the following code?

```
try:
    name = person.name
except NameError:
    print("Error: NameError - variable 'person' is not defined")
```

a) Error: NameError - variable 'person' is not defined
b) NameError
c) Error: 'NoneType' object has no attribute 'name'
d) None of the above

206. What is the output of the following code?

```
try:
    num = int("10.5")
except ValueError:
    print("Error: Invalid literal for int()")
else:
    print("Number:", num)
finally:
    print("Finally block executed")
```

a) Error: Invalid literal for int()
   Finally block executed
b) Number: 10
   Finally block executed
c) ValueError
   Finally block executed
d) None of the above

207. What is the output of the following code?

```
try:
    num = int(input("Enter a number: ")) # input="FAI"
except ValueError:
    print("Error: Invalid input. Please enter a valid number.")
except EOFError:
    print("Error: End of input reached unexpectedly.")
```

a) Error: Invalid input. Please enter a valid number.
b) Error: End of input reached unexpectedly.
c) ValueError
d) None of the above

208. What is the output of the following code?

```
try:
    file = open("nonexistent.txt", "r")
except IOError:
    print("Error: Input/Output error occurred")
```

a) IOError
b) Error: Input/Output error occurred
c) Error: 'module' object has no attribute 'open'
d) None of the above

209. What is the output of the following code?

```
try:
    raise StopIteration("Iteration stopped")
except StopIteration as e:
    print("Error:", str(e))
```

a) Iteration stopped
b) StopIteration
c) Error: 'NoneType' object has no attribute 'message'
d) Error: Iteration stopped

210. What is the output of the following code?

```
try:
    assert 2 + 2 == 5, "Assertion failed"
except AssertionError as e:
    print("Error:", str(e))
```

a) Error: Assertion failed
b) AssertionError
c) Error: 'NoneType' object has no attribute 'message'
d) None of the above

211. What is the output of the following code?

```
try:
    import non_existent_module
except ImportError:
    print("Error: Failed to import the module")
```

a) Error: Failed to import the module
b) ImportError
c) Error: 'NoneType' object has no attribute 'message'
d) None of the above

212. What is the output of the following code?

```
try:
    num = int("abc")
except ValueError:
    print("Error: Invalid literal for int()")
except Exception:
    print("Error: Something went wrong")
```

a) Error: Something went wrong
b) ValueError
c) Error: Invalid literal for int()
d) None of the above

213. What is the output of the following code?

```
try:
    num = int(10.5)
except ValueError:
    print("Error: Invalid literal for int()")
except Exception:
    print("Error: Something went wrong")
else:
    print("Number:", num)
finally:
    print("Finally block executed")
```

a) Error: Invalid literal for int()
b) Number: 10.5
   Finally block execute
c) ValueError
   Finally block executed
d) Number: 10
   Finally block executed

214. What is the output of the following code?

```
try:
    lst = [1, 2, 3]
    iterator = iter(lst)
    while True:
        print(next(iterator))
except StopIteration:
    print("Error: End of iteration reached")
```

a) Error: End of iteration reached
b) 1, 2, 3, Error: End of iteration reached
c) StopIteration
d) None of the above

Modules:

215. Which keyword is used to import a module in Python?
a) import
b) load
c) require
d) include

216. Which module is commonly used for working with dates and times in Python?
a) datetime
b) time
c) calendar
d) math

217. What does the "as" keyword do when importing a module?
a) Disables the module
b) Imports all functions from the module
c) Renames the module
d) None of the above

218. What is the correct way to import only the "sum" function from the "math" module?
    a) import math.sum
    b) import sum from math
    c) from math import sum
    d) from math import *

219. Which module can be used to generate random numbers in Python?
    a) random
    b) math
    c) statistics
    d) numpy

220. What will be the output of the following code snippet?

```
from random import randint
print(randint(1, 10))
```

    a) A random number between 1 and 10
    b) Error: undefined function randint()
    c) 1
    d) 10

221. Which module is commonly used for handling file input/output operations in Python?
    a) sys
    b) fileio
    c) io
    d) os

222. What is the purpose of the "sys" module in Python?
    a) Access system-specific parameters and functions
    b) Manipulate files and directories
    c) Perform mathematical operations
    d) None of the above

223. What will be the output of the following code snippet?

```
import os  # The current running directory is "C:\Users\ThrOw"
print(os.getcwd())
```

a) Error: undefined function getcwd()
b) C:\Users\ThrOw
c) Sorry use the CMD
d) The root directory

224. Given the following code snippet, what will be the output?

```
from datetime import datetime
print(datetime.now().year)
```

a) The current year (2023)
b) Error: undefined attribute now()
c) The current date and time
d) The current month

225. What is the output of the following code snippet?

```
import statistics
numbers = [2, 4, 6, 8, 10]
mean = statistics.mean(numbers)
print(round(mean, 2))
```

a) 2
b) 6
c) 8
d) 5

226. Given the following code snippet, what will be the output?

```
import math
angle = math.radians(45)
print(math.sin(angle))
```

a) 0.5
b) 1.0
c) 0.707
d) Error: undefined function radians()

227. What will be the output of the following code snippet?

```
import calendar
print(calendar.isleap(2024))
```

a) True
b) False
c) Error: undefined function isleap()
d) 2024

228. Given the following code snippet, what will be the output?

```
import sys
print(sys.platform)
```

a) The current operating system
b) Error: undefined attribute platform()
c) The current Python version
d) The current system architecture

229. What is the output of the following code snippet?

```
from urllib.request import urlopen
response = urlopen("https://www.example.com")
print(response.getcode())
```

a) The HTTP response code of the URL
b) Error: undefined function getcode()
c) The content of the webpage
d) The status of the URL connection

230. Given the following code snippet, what will be the output?

```
import numpy as np

matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
result = np.transpose(matrix)

print(result)
```

a) [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
b) [[1, 4, 7], [2, 5, 8], [3, 6, 9]]
c) [[1, 2, 3], [7, 8, 9], [4, 5, 6]]
d) [[3, 6, 9], [2, 5, 8], [1, 4, 7]]

231. Given the following code snippet, what will be the output?

```
import numpy as np
numbers = np.array([1, 2, 3, 4, 5])
print(np.mean(numbers))
```

a) 3.0
b) Error: undefined function mean()
c) [1, 2, 3, 4, 5]
d) Error: Failed to import the module

232. What will be the output of the following code snippets:

```
# file: module_a.py
import module_b

def greet():
    return "Hello!"

print(module_b.calculate(5, 10))
print(greet())
```

```
# file: module_b.py
def calculate(a, b):
    return a * b

print("Module B imported!")
```

What will be the output when module_a.py is executed?

a) Module B imported!, Hello!, 50
b) Hello!, 50, Module B imported!
c) Module B imported!, 50, Hello!
d) Hello!, Module B imported!, 50

233. Consider the following code snippets:

```
# file: a.py
def add(x, y):
    return x * y
```

```
# file: b.py
import a

def subtract(x, y):
    return a.add(x, y)

def add(a, b):
    return x, y
```

What will be the output when this code is executed?

```
import b
result = b.subtract(10, 5)
print(result)
```

a) **5**
b) **15**
c) **50**
d) **Error: undefined function subtract()**

234. Consider the following code snippets:

```
# file: module_a.py
from module_b import multiply

def square(x):
    return multiply(x, x)

result = square(4)
print(result)
```

```
# file: module_b.py
from module_a import square

def multiply(a, b):
    return a * b

print(multiply(5, 3))
print(square(3))
```

What will be the output when module_a.py is executed?

a) **9, 15, 16**
b) **9, 15, Error: undefined function square()**
c) **16, 15, 9**
d) **Error: circular import between module_a and module_b**

235. What will be the output of the following code snippets?

```
# file: module_b.py
from module_c import subtract

def divide(a, b):
    return subtract(a, b)

def subtract(a, b):
    return a - b
```

```
# file: module_c.py
def subtract(a, b):
    return a - b
```

```
from module_b import divide

result = divide(10, 2)
print(result)
```

a) 8
b) 5
c) Error: undefined function divide()
d) Error: undefined function subtract()

236. Consider the following code snippets:

```
# file: module_b.py
import module_c

def add_numbers(a, b):
    return module_c.multiply_numbers(a, b)
```

```
# file: module_c.py
def multiply_numbers(a, b):
    return a * b
```

```
# file: module_c.py
import module_b

result = module_b.add_numbers(3, 4)
print(result)
```

What will be the output when module_a.py is executed?

a) **7**
b) **12**
c) **Error: undefined function add_numbers()**
d) **Error: undefined function multiply_numbers()**

237. Consider the following code snippets:

```python
# file: module_a.py
from module_b import MyClass

obj = MyClass()
result = obj.add_numbers(3, 4)
print(result)
```

```python
# file: module_b.py
class MyClass:
    def add_numbers(self, a, b):
        return str(a) + str(b)
```

What will be the output when module_a.py is executed?

a) **7**
b) **Error: undefined class MyClass**
c) **34**
d) **Error: undefined method add_numbers()**

238. Consider the following code snippets:

```python
# file: module_a.py
from module_b import Circle

circle = Circle(5)
area = circle.calculate_area()
print(area)
```

```python
# file: module_b.py
import math

class Circle:
    def __init__(self, radius):
        self.radius = radius

    def calculate_area(self):
        return math.pi * (self.radius ** 2)
```

What will be the output when module_a.py is executed?

a) **8.54**
b) **Error: undefined class Circle**
c) **25**
d) **78.54**

239. What will be the output of the following code snippets:

```
# file: module_a.py
from module_b import Rectangle

rectangle = Rectangle(4, 6)
perimeter = rectangle.calculate_perimeter()
print(perimeter)
```

```
# file: module_b.py
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def calculate_perimeter(self):
        return 2 * (self.length + self.width)
```

What will be the output when module_a.py is executed?

a) **20**
b) **Error: undefined class Rectangle**
c) **24**
d) **Error: undefined method calculate_perimeter()**

240. What will be the output of the following code snippets:

```
# file: module_a.py
from module_b import Car

car = Car("Tesla", "Model S")
car.start_engine()
```

```python
# file: module_b.py
class Car:
    def __init__(self, make, model):
        self.make = make
        self.model = model
        self.engine = Engine()

    def start_engine(self):
        self.engine.start()

class Engine:
    def start(self):
        print("Engine started.")
```

a) Engine started.
b) "Tesla", "Model S"
c) Error: undefined class Engine
d) Error: undefined method start_engine()

241. **What will be the output of the following code snippets:**

```python
# file: module_a.py
from module_b import Square

square = Square(5)
area = square.calculate_area()
print(area)
```

```python
# file: module_b.py

class Square():
    def __init__(self, side_length):
        self.side_length = side_length

    def calculate_area(self):
        return "Error: undefined class Square
```

What will be the output when module_a.py is executed?

a) 10
b) Error: undefined class Square
c) 25
d) Error: undefined method calculate_area()

242. What will be the output of the following code:

```
from math import sqrt

def sqrt(n):
    return "Hello"

result = sqrt(16)
print(result)
```

a) 4
b) "Hello"
c) Error: undefined function sqrt()
d) Error: incompatible types in sqrt()

243. What will be the output of the following code:

```
import datetime

datetime.datetime = None

today = datetime.date.today()
print(today)
```

a) The current date
b) Error: undefined function date()
c) None
d) Error: incompatible assignment to datetime module

244. What will be the output of the following code:

```
from random import randint

def randint(a, b):
    return a + b

result = randint(1, 10)
print(result)
```

a) A random number between 1 and 10
b) Error: undefined function randint()
c) 11
d) Error: incompatible types in randint()

245. **What will be the output of the following code:**

```
import statistics

statistics.mean = lambda x: sum(x)

numbers = [1, 2, 3, 4, 5]
result = statistics.mean(numbers)
print(result)
```

a) **3**
b) **Error: undefined function mean()**
c) **15**
d) **Error: incompatible assignment to statistics module**

246. **What will be the output of the following code:**

```
import numpy as np


matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

result = np.linalg.det(matrix)
result2 = np.linalg.inv(matrix)

print(result)
print(result2)
```

What does the above code snippet compute?

a) **The inverse of the given matrix, then its determinant**
b) **The determinant of the given matrix, then its inverse**
c) **The trace of the given matrix**
d) **The eigenvalues of the given matrix**

247. **Consider the following code snippets:**

```
# file: module_a.py
import numpy as np
from module_b import get_eigenvalues_eigenvectors

matrix = np.array([[2, -1], [4, 3]])

eigenvalues, eigenvectors = get_eigenvalues_eigenvectors(matrix)

print("Eigenvalues:", eigenvalues)
print("Eigenvectors:", eigenvectors)
```

```python
# file: module_b.py
import numpy as np

def get_eigenvalues_eigenvectors(matrix):
    eigenvalues, eigenvectors = np.linalg.eig(matrix)
    return eigenvalues, eigenvectors
```

What will be the output when module_a.py is executed?

a) **Eigenvalues: [2.5+1.93649167j 2.5-1.93649167j]**
Eigenvectors: [[-0.1118034 +0.4330127j -0.1118034 -0.4330127j]
[0.89442719+0.j 0.89442719-0.j]]

b) **Eigenvalues: [2+1.73205081j 2-1.73205081j]**
Eigenvectors: [[-0.57735027+0.j -0.57735027-0.j]
[0.57735027+0.81649658j 0.57735027-0.81649658j]]

c) **Eigenvalues: [3+2j 3-2j]**
Eigenvectors: [[0.70710678+0.j 0.70710678-0.j]
[0.70710678+0.j 0.70710678-0.j]]

d) **Eigenvalues: [2+0j 3+0j]**
Eigenvectors: [[-0.89442719+0.j -0.89442719-0.j]
[0.4472136 +0.j 0.4472136 -0.j]]

248. What will be the output of the following code:

```python
import random

random.choice = lambda x: x[2]

items = [1, 2, 3, 4, 5]
result = random.choice(items)
print(result)
```

a) **[1, 2, 3, 4, 5]**
b) **Error: undefined function choice()**
c) **2**
d) **3**

249. Consider the following code snippets

```python
# file: module_b.py
def dot_product(vector_a, vector_b):
    transposed_b = transpose(vector_b)
    result = 0
    for a, b_list in zip(vector_a, transposed_b):
        for b in b_list:
            result += a * b
    return result

def transpose(vector):
    if isinstance(vector, list):  # Check if vector is a list
        if isinstance(vector[0], list):  # Check if vector contains sublists
            return list(map(list, zip(*vector)))
        else:
            return [[x] for x in vector]  # Convert single vector to a list of
single-element vectors
    else:
        raise TypeError("Input vector must be a list.")
```

```python
# file: module_a.py
from module_b import dot_product

vector_a = [2, 4, 6]
vector_b = [1, 3, 5]
result = dot_product(vector_a, vector_b)
print(result)
```

What will be the output when module_a.py is executed

a) **44**
b) **56**
c) **Error: undefined function dot_product()**
d) **Error: cannot import name 'transpose' from 'module_c'**

250. Consider the following code snippets:

```
# file: module_a.py
class MyClass:
    def __init__(self, value):
        self.value = value

    def __add__(self, other):
        return self.value * other
```

```
# file: module_b.py
from module_a import MyClass

obj_b = MyClass(5)
result = obj_b + 3
print(result)
```

a) **8**
b) **15**
c) **5**
d) **3**

With my best wishes

Follow me