Python OOP Assignment

Q1. What is the purpose of Python's OOP?

A-1-The purpose of Python's Object-Oriented Programming (OOP) is to organize code into reusable, modular structures called objects that can interact with each other to perform tasks. OOP allows developers to write code that models real-world objects or abstract concepts as classes, and create instances of those classes as objects that have their own unique properties and methods.
OOP in Python provides many benefits such as code reusability, encapsulation, abstraction, and inheritance. By using OOP, developers can create more organized, maintainable, and scalable code, which is essential for building complex software systems. OOP in Python also enables developers to write code that is easier to understand, debug, and test, making it a popular choice for large-scale software development projects.

Q2. Where does an inheritance search look for an attribute?

A-2-In Python, when an attribute is accessed on an instance of a class, the inheritance search first looks for the attribute within the instance itself. If the attribute is not found within the instance, the search then proceeds to the class of the instance, and then to its parent classes in the order specified by the method resolution order (MRO).

Q3. How do you distinguish between a class object and an instance object?

A-3-A class object is an instance of the type metaclass, which is responsible for creating and defining classes. A class object defines the blueprint for creating instances of that class, and contains attributes and methods that are shared by all instances of the class.On the other hand, an instance object is an individual object created from a class. Each instance of a class has its own set of attributes and methods that are unique to that instance.

Q4. What makes the first argument in a class’s method function special?

A-4-the first argument of a class's method function is conven-
-tionally named self, and it refers to the instance of the class that the method is being called on. This argument is

special because it allows the method to access and manipulate the instance's data and attributes.
for ex-

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

Q5. What is the purpose of the init method?

A-5-the __init__() method or constructor is a special method that is called when an instance of a class is created. The purpose of the __init__() method is to initialize the instance's attributes with the values provided during instantiation.
The __init__() method takes the self argument, which refers to the instance being created, and any other arguments that are required to initialize the instance's attributes. The self argument is always the first argument in the method definition and is automatically passed by Python when the method is called.
for ex-

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

Q6. What is the process for creating a class instance?

A-6- the process of creating a class instance is :

```
# Define the class
class MyClass:
    def __init__(self, x, y):
        self.x = x
        self.y = y

# Create an instance of the class
my_obj = MyClass(10, 20)

# Access the instance's attributes
print(my_obj.x)  # Output: 10
print(my_obj.y)  # Output: 20
```

Q7. What is the process for creating a class?

A-7-

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

    def get_name(self):
        return self.name

    def get_age(self):
        return self.age

# Create an instance of the class
person = Person("Alice", 25)

# Use the methods of the class
print(person.get_name())  # Output: Alice
print(person.get_age())   # Output: 25
```

Q8. How would you define the superclasses of a class?

A-8-

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

  def info(self):
    print(f"{self.name} age is {self.age}.\n")

class employee(person):
  def skill(self):
    print("his expertise in python")


a=employee("rishabh","26")
a.info() #output:rishabh age is 26.
a.skill() #output:his expertise in python
```

In the above ex person is superclass.

Q9. What is the relationship between classes and modules?

A-9-classes and modules are related in that they are both used

to organize code and provide abstraction and encapsulation.
a module can contain one or more classes, which can be used by
other modules or programs that import the module. This allows
for code reuse and modularity.

Q10. How do you make instances and classes?

A-10-We can create instances and classes using the class keyword
and the __init__() method(constructor).
for ex-

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

    def say_hello(self):
        print(f"Hello, {self.name}!")
```

Q11. Where and how should be class attributes created?

A-11-class attributes are variables that are shared by all
instances of a class. They are defined inside the class but
outside of any method. Class attributes are accessed using
the class name rather than an instance of the class.
for ex-

```
class employee:
  company_name="Tesla"
  def __init__(self,name):
    self.name = name
```

Q12. Where and how are instance attributes created?

A-12-Instance attributes are variables that are specific to an
instance of a class. They are defined inside the __init__()
method of a class and are accessed using the self keyword.
for ex-

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

Q13. What does the term "self" in a Python class mean?

A-13-In Python, the term self is a reference to the instance of a class that a method is being called on. It's a convention in Python to call the first parameter of instance methods self, although you can technically call it whatever you want.

Q14. How does a Python class handle operator overloading?

A-14-In Python, operator overloading is the ability to define the behavior of operators such as +, -, *, /, and others for instances of a class. To overload an operator, you need to define a method with a special name that corresponds to the operator.
for ex-

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

    def __add__(self, other):
        return MyClass(self.value + other.value)
a = MyClass(1)
b = MyClass(2)
c = a + b
print(c.value)  # Output: 3
```

Q15. When do you consider allowing operator overloading of your classes?

A-15-you should consider allowing operator overloading of your classes in Python when it makes sense logically or mathematically, but you should be cautious and ensure that the overloaded operators behave consistently and intuitively, and that they don't make your code less readable or understandable.

Q16. What is the most popular form of operator overloading?

A-16-the most popular form of operator overloading is probably the __add__() method, which is used to overload the + operator. This is because addition is a fundamental operation in many mathematical and computational contexts, and overloading the + operator can make code more concise, readable, and expressive.

Q17. What are the two most important concepts to grasp in order to comprehend Python OOP code?

A-17-The two most important concepts to grasp in order to comprehend Python OOP code are:

*Classes: A class is a blueprint for creating objects, which encapsulate data and behavior. It defines the properties (attributes) and methods that are common to all instances of the class. Classes are used to create new instances (objects) of that class, which can be manipulated and modified independently.

*Objects: An object is an instance of a class. It represents a specific realization of the class blueprint, with its own data and behavior. Objects can be created from a class using the constructor method (__init__()), and can be accessed and manipulated through their attributes and methods.

Q18. Describe three applications for exception processing.

A-18-three common applications for exception processing in Python:

*Input validation: When you're writing a program that accepts user input, it's important to validate that input to ensure that it meets certain requirements.
*Resource management: In many programs, you need to manage resources such as files, network connections, or database connections.
*Error reporting: When an error occurs in a program, By using exception processing to catch errors and generate informative error messages, you can help users understand what's going wrong and how to fix it.

Q19. What happens if you don't do something extra to treat an exception?

A-19-If an exception occurs in Python and you don't handle it with an appropriate exception handling code block, then the Python interpreter will terminate your program and print a traceback error message to the console.This means that any code that comes after the exception-causing code will not be executed, and your program will exit prematurely. This can lead to unexpected behavior and bugs in your program.

Q20. What are your options for recovering from an exception in your script?

A-20-When an exception occurs in your Python script,

we can recover from it by using one or more of the
following options:

*Catch the exception using a try-except block
*Raise a new exception
*Retry the operation

Q21. Describe two methods for triggering exceptions in your script.

A-21-two methods for triggering exceptions in your script-

*Raise an exception manually: You can manually trigger
an exception in your Python script by using the raise
statement.
*Call a function that raises an exception: You can also
trigger an exception in your Python script by calling
a function that is designed to raise an exception in
certain circumstances.

Q22. Identify two methods for specifying actions to be executed at termination time, regardless of
whether or not an exception exists.

A-22-
*The finally block: You can use a finally block in a
try-except-finally statement to specify actions that
should be executed regardless of whether an exception
occurs or not.
*The atexit module: The atexit module provides a way
to register functions that should be called when a
Python program is about to exit, either normally
or due to an unhandled exception.

Q23. What is the purpose of the try statement?

A-23-The purpose of the try statement in Python is
to handle exceptions that may occur during the
execution of a block of code.

Q24. What are the two most popular try statement variations?

A-24-
*try-except: This variation allows you to catch and handle specific types of exceptions that may occur within the try block
for ex-

```
try:
    # some code that may raise an exception
except SomeException:
    # handle the exception
```

*try-finally: This variation allows you to specify cleanup code that should be executed regardless of whether an exception occurs in the try block or not.
for ex-

```
try:
    # some code that may raise an exception
finally:
    # cleanup code that should be executed whether an exception occurred or not
```

Q25. What is the purpose of the raise statement?

A-25-The purpose of the raise statement in Python is to raise an exception explicitly. When you use the raise statement, you are instructing the Python interpreter to stop executing the current block of code and look for an exception handler further up the call stack.
for ex-

```
raise SomeException("An error message")
```

Q26. What does the assert statement do, and what other statement is it like?

A-26-The assert statement in Python is used as a debugging aid to check that a condition is true. It is similar to the if statement, but is used specifically for debugging purposes.
for ex-assert condition, message

Q27. What is the purpose of the with/as argument, and what other statement is it like?

A-27-The with/as statement in Python is used to simplify the management of resources that need to be explicitly

opened and closed, such as files, network connections, and database connections. It ensures that the resource is properly closed when the block of code enclosed by the with statement is exited, even if an exception occurs.
for ex-

```
with open('filename.txt', 'r') as file:
    # some code that uses the file object
```

The with/as statement is similar to the try/finally statement in that both are used to manage resources that need to be explicitly opened and closed.

Q28. What are *args, **kwargs?

A-28-*args is used to pass a variable number of positional arguments to a function. It allows you to pass an arbitrary number of arguments to a function, which are then collected into a tuple.

**kwargs is used to pass a variable number of keyword arguments to a function. It allows you to pass an arbitrary number of keyword arguments to a function, which are then collected into a dictionary.

Q29. How can I pass optional or keyword parameters from one function to another?

A-29-It can be done by using arguments as *args ,**kwargs.

Q30. What are Lambda Functions?

A-30-Lambda functions, also known as anonymous functions, are a type of function in Python that are defined using a single line of code and do not have a name. Lambda functions are often used as a shorthand way of defining small, simple functions that are only needed once in a program.
for ex-

```
double=lambda x:x*2
```

```
print(double)
```

Q31. Explain Inheritance in Python with an example?

A-31-Inheritance is a fundamental concept in object-oriented programming that allows you to define a new class based on an existing class.

Types of inheritance:

*Single Inheritance:Single inheritance enables a derived class to inherit properties from a single parent class.
ex-

```
class Parent:
    def func1(self):
        print("This function is in parent class.")

class Child(Parent):
    def func2(self):
        print("This function is in child class.")
```

*Multiple Inheritance:When a class can be derived from more than one base class this type of inheritance is called multiple inheritances.
ex-

```
class Mother:
    mothername = ""

    def mother(self):
        print(self.mothername)
class Father:
    fathername = ""
    def father(self):
        print(self.fathername)
class Son(Mother, Father):
    def parents(self):
        print("Father name is :", self.fathername)
        print("Mother :", self.mothername)
```

*Multilevel Inheritance:In multilevel inheritance, features of the base class and the derived class are further inherited into the new derived class.
ex-

```
class Grandfather:
     def __init__(self, grandfathername):
        self.grandfathername = grandfathername
```

```
class Father(Grandfather):
    def __init__(self, fathername, grandfathername):
        self.fathername = fathername
        Grandfather.__init__(self, grandfathername)
class Son(Father):
    def __init__(self, sonname, fathername, grandfathername):
        self.sonname = sonname
        Father.__init__(self, fathername, grandfathername)
     def print_name(self):
        print('Grandfather name :', self.grandfathername)
        print("Father name :", self.fathername)
        print("Son name :", self.sonname)
```

*Hierarchical Inheritance:When more than one derived class are created from a single base this type of inheritance is called hierarchical inheritance.

ex-

```
class Parent:
    def func1(self):
        print("This function is in parent class.")
class Child1(Parent):
    def func2(self):
        print("This function is in child 1.")
class Child2(Parent):
    def func3(self):
        print("This function is in child 2.")
```

*Hybrid Inheritance:Inheritance consisting of multiple types of inheritance is called hybrid inheritance.

ex-

```
class School:
    def func1(self):
        print("This function is in school.")
class Student1(School):
    def func2(self):
        print("This function is in student 1. ")
class Student2(School):
    def func3(self):
        print("This function is in student 2.")
class Student3(Student1, School):
    def func4(self):
        print("This function is in student 3.")
```

Q32. Suppose class C inherits from classes A and B as

class C(A,B).Classes A and B both have their own versions of method func(). If we call func() from an object of class C, which version gets invoked?

A-32-the method func() of class A will be used if both classes A and B have their own versions of func().

Q33. Which methods/functions do we use to determine the type of instance and inheritance?

A-33-type(),issubclass() and isinstance() functions can be used.

Q34.Explain the use of the 'nonlocal' keyword in Python.

A-34-The nonlocal keyword in Python is used within a nested function to access and modify a variable that is defined in the enclosing function.

Q35. What is the global keyword?

A-35-the global keyword is used to declare a variable as global inside a function, allowing the function to access and modify a variable that is defined outside of its own scope.