1. Write what is meant by operator overloading and method overriding with examples.

Operator overaloading:

Operator Overloading means giving extended meaning beyond their predefined operational meaning. For example operator + is used to add two integers as well as join two strings and merge two lists. It is achievable because '+' operator is overloaded by int class and str class. You might have noticed that the same built-in operator or function shows different behavior for objects of different classes, this is called *Operator Overloading*.

Consider that we have two objects which are a physical representation of a class (user-defined data type) and we have to add two objects with binary '+' operator it throws an error, because compiler don't know how to add two objects. So we define a method for an operator and that process is called operator overloading.

Example:

```
# Python program to overload equality
# and less than operators

class A:
        def __init__(self, a):
                self.a = a
        def __lt__(self, other):
                if(self.a<other.a):
                        return "ob1 is lessthan ob2"
                else:
                        return "ob2 is less than ob1"
```

```
    def __eq__(self, other):
        if(self.a == other.a):
            return "Both are equal"
        else:
            return "Not equal"

ob1 = A(2)
ob2 = A(3)
print(ob1 < ob2)

ob3 = A(4)
ob4 = A(4)
print(ob1 == ob2)
```

Method overriding:

Method overriding is an ability of any object-oriented programming language that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.

Example:

```
# method overriding

# Defining parent class
class Parent():

    # Constructor
    def __init__(self):
```

```
            self.value = "Inside Parent"

        # Parent's show method
        def show(self):
            print(self.value)

# Defining child class
class Child(Parent):

        # Constructor
        def __init__(self):
            self.value = "Inside Child"

        # Child's show method
        def show(self):
            print(self.value)



# Driver's code
obj1 = Parent()
obj2 = Child()

obj1.show()
obj2.show()
```

Output:

```
Inside Parent
Inside Child
```

```
class Person:
    def __init__(self, name, age, height, weight) -> None:
```

```
        self.name = name
        self.age = age
        self.height = height
        self.weight = weight

    class Cricketer(Person):
       def __init__(self, name, age, height, weight) -> None:
          super().__init__(name, age, height, weight)

    sakib = Cricketer('Sakib', 38, 68, 91)
    musfiq = Cricketer('Rahim', 36, 68, 88)
    kamal = Cricketer('Kamal', 39, 68, 94)
    jack = Cricketer('Jack', 38, 68, 91)
    kalam = Cricketer('Kalam', 37, 68, 95)
```

2. Find out which of these players is older using the operator overloading.

Ans:

```
class Person:
    def __init__(self, name, age, height, weight) -> None:
        self.name = name
        self.age = age
        self.height = height
        self.weight = weight

class Cricketer(Person):
    def __init__(self, name, age, height, weight) -> None:
        super().__init__(name, age, height, weight)

    def __gt__(self, other):
        return self.age > other.age

# Create Cricketer instances
sakib = Cricketer('Sakib', 38, 68, 91)
musfiq = Cricketer('Rahim', 36, 68, 88)
kamal = Cricketer('Kamal', 39, 68, 94)
jack = Cricketer('Jack', 38, 68, 91)
kalam = Cricketer('Kalam', 37, 68, 95)
```

```
# Find the oldest cricketer
cricketers = [sakib, musfiq, kamal, jack, kalam]
oldest = max(cricketers)

print(f"The oldest cricketer is {oldest.name} with an age of
{oldest.age}.")
```

3. Write down 4 differences between the class method and static method with proper examples.

Ans:

Class method vs Static Method

The difference between the Class method and the static method is:

- A class method takes cls as the first parameter while a static method needs no specific parameters.
- A class method can access or modify the class state while a static method can't access or modify it.
- In general, static methods know nothing about the class state. They are utility-type methods that take some parameters and work upon those parameters. On the other hand class methods must have class as a parameter.
- We use @classmethod decorator in python to create a class method and we use @staticmethod decorator to create a static method in python.

When to use the class or static method?

- We generally use the class method to create factory methods. Factory methods return class objects (similar to a constructor) for different use cases.
- We generally use static methods to create utility functions.

Aspect	Class Method	Static Method	Instance Method
Definition	A method bound to the class itself, defined with “@classmethod”	A method that does not receive an implicit first argument (‘self’ or ‘cls’), defined with “@staticmethod”	A method defined within a class, taking `self` as the first parameter, representing the instance.
Access	Can access class attributes and modify them.	Cannot access or modify class or instance attributes.	Can access and modify

			instance attributes.
First Argument	Receives ‘cls’ as the first argument representing the class.	No implicit first argument is passed to the method.	Receives `self` as the first argument representing the instance.
Usage	Often used for operations that modify or interact with class-level data.	Typically used for utility functions that don’t depend on instance or class state.	Commonly used for operations specific to individual instances.
Inheritance	Can be overridden in subclasses, with ‘cls’ referring to the subclass.	Can be called directly via the class or subclass, but not overridden.	Can be overridden in subclasses, with ‘self’ referring to the

			subclass instance.
Access Modifier	Typically used when the method needs access to class-level data.	Suitable when the method doesn't rely on instance or class attributes.	Preferred when the method operates on instance-specific attributes.
Example Use	Calculating statistics across all instances of a class.	Formatting dates, performing mathematical operations.	Returning information specific to an instance, like its attributes.
Usefulness	Useful when dealing with class-level functionality and shared attributes.	Handy for standalone functions related to the class but not dependent on its state.	Essential for modifying and accessing instance attributes and behaviors.

#class method

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

	def get_value(self):
		return self.value

# Create an instance of MyClass
obj = MyClass(10)

# Call the get_value method on the instance
print(obj.get_value())
 # Output: 10
```

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

	@staticmethod
	def get_max_value(x, y):
		return max(x, y)

# Create an instance of MyClass
obj = MyClass(10)

print(MyClass.get_max_value(20, 30))

print(obj.get_max_value(20, 30))
```

Output

```
30
30
```

4. Write what are getter and setter with proper examples

```
class Label:
    def __init__(self, text, font):
        self._text = text
        self._font = font

    def get_text(self):
        return self._text

    def set_text(self, value):
        self._text = value

    def get_font(self):
        return self._font

    def set_font(self, value):
        self._font = value
```

5. Explain the difference between inheritance and composition with proper examples.