**Q1. Which two operator overloading methods can you use in your classes to support iteration?**

* **\_\_iter\_\_**:
  + Returns an iterator object.
  + Example:

class MyIterable:

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

self.data = data

def \_\_iter\_\_(self):

return iter(self.data)

* **\_\_next\_\_**:
  + Defines the logic for retrieving the next item in the iteration.
  + Example:

class MyIterator:

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

self.data = data

self.index = 0

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

if self.index >= len(self.data):

raise StopIteration

value = self.data[self.index]

self.index += 1

return value

**Q2. In what contexts do the two operator overloading methods manage printing?**

* **\_\_str\_\_**:
  + Defines the "informal" string representation of an object.
  + Used by print() and str().
  + Example:

class MyClass:

def \_\_str\_\_(self):

return "This is MyClass"

* **\_\_repr\_\_**:
  + Defines the "official" string representation of an object.
  + Used by repr() and in the interactive interpreter.
  + Example:

class MyClass:

def \_\_repr\_\_(self):

return "MyClass()"

**Q3. In a class, how do you intercept slice operations?**

* **\_\_getitem\_\_**:
  + Intercepts slice operations when accessing items using square brackets ([]).
  + Example:

class MyClass:

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

self.data = data

def \_\_getitem\_\_(self, index):

return self.data[index]

**Q4. In a class, how do you capture in-place addition?**

* **\_\_iadd\_\_**:
  + Intercepts the += operator for in-place addition.
  + Example:

class MyClass:

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

self.value = value

def \_\_iadd\_\_(self, other):

self.value += other

return self

**Q5. When is it appropriate to use operator overloading?**

* **Appropriate Use Cases**:
  + When the class represents a mathematical or logical entity (e.g., vectors, matrices).
  + When the operators have a clear and intuitive meaning for the class.
  + When it improves code readability and expressiveness.
* **Example**:
  + Overloading + for a Vector class:

class Vector:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def \_\_add\_\_(self, other):

return Vector(self.x + other.x, self.y + other.y)