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

**Answer**: The \_\_iter\_\_ & the \_\_next\_\_. The \_\_iter\_\_ returns the iterator object and is implicitly called at the start of loops. The \_\_next\_\_ method returns the next value and is implicitly called at each loop increment.

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

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

**Answer**: Python slice() function returns a slice object. A sequence of objects of any type(string, bytes, tuple, list or range) or the object which implements \_\_getitem\_\_() and \_\_len\_\_() method then this object can be sliced using slice() method.

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

**Answer**: Python provides the operator x += y to add two objects in-place by calculating the sum x + y and assigning the result to the first operands variable name x. You can set up the in-place addition behaviour for your own class by overriding the magic “dunder” method \_\_iadd\_\_(self, other) in your class definition.

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

**Answer**: 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. We can overload all existing operators but we can’t create a new operator.

To perform operator overloading, Python provides some special function or magic function that is automatically invoked when it is associated with that particular operator. For example, when we use + operator, the magic method \_\_add\_\_ is automatically invoked in which the operation for + operator is defined.