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

ANS) With operator overloading, we are able to change the meaning of a Python operator within the scope of a class.

|  |  |
| --- | --- |
| + | \_\_add\_\_(self, other) |
| - | \_\_sub\_\_(self, other) |

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

ANS) You can define the string format your object should be displayed in when passed to str() by defining the \_\_str\_\_() method in your class. Moreover, \_\_str\_\_() is the method that is used by Python when you call print() on your object. It is necessary that \_\_str\_\_() returns a str object, and we get a TypeError if the return type is non-string

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

ANS) First, we get the slice object that is passed in to \_\_getitem\_\_ and use it to slice the dictionary's key values. Then, using dictionary comprehension, we put all of the sliced keys and values into a new dictionary using our sliced keys. This gives us a nice sliced dictionary that we can use for whatever we like.

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

ANS) Normal operator's “add()” method, implements “a+b” and stores the result in the mentioned variable. Inplace operator's “iadd()” method, implements “a+=b” if it exists (i.e in case of immutable targets, it doesn't exist) and changes the value of passed argument. But if not, “a+b” is implemented.

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

ANS) When you pass an instance of some class to a built-in function or use an operator on the instance, it is actually equivalent to calling a special method with relevant arguments.