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

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. \_\_next\_\_ raises a StopIteration exception when there are no more value to return, which is implicitly captured by looping constructs to stop iterating.

class Counter:

def \_\_init\_\_(self, low, high):

self.current = low

self.high = high

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

return self

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

if self.current > self.high:

raise StopIteration

else:

self.current += 1

return self.current - 1

for num in Counter(5, 15):

print(num)

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

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. \_\_next\_\_ raises a StopIteration exception when there are no more value to return, which is implicitly captured by looping constructs to stop iterating.

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

The \_\_getitem\_\_ method is used for accessing list items, array elements, dictionary entries etc. slice is a constructor in Python that creates slice object to represent set of indices that the range(start, stop, step) specifies. \_\_getitem\_\_ method can be implement in a class, and the behavior of slicing can be defined inside it.

Syntax: \_\_getitem\_\_(slice(start, stop, step))

Parameter:

slice() : constructor to create slice object.

start: An integer number specifying start index.It is optional and default is 0.

stop: An integer number specifying end index.

step: An integer number specifying the step of slicing. It is optional and

default is 1.

Code:

sliced ='abcde'.\_\_getitem\_\_(slice(0, 2, 1))

print(sliced)

The string abcde is sliced with starting index 0 and stop index 2 with step index 1 hence it slices ab from the string and prints the output.

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

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 behavior 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?

When one or both operands are of a user-defined class or structure type, operator overloading makes it easier to specify user-defined implementation for such operations. This makes user-defined types more similar to the basic primitive data types in terms of behaviour.

It allows for reusability; instead of developing numerous methods with minor differences, we can simply write one method and overload it. It also increases code clarity and reduces complexity. Operator overloading also makes the code more concise and easier to understand.

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.