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

\_\_iter\_\_ Method: Defines what is returned as an iterator for the class, initializing any necessary state.

\_\_next\_\_ Method: Defines how the next item is produced and when to stop iteration.

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

\_\_str\_\_ Method: Manages informal, user-friendly string representation of the object for end-user output. Used by print() and str().

\_\_repr\_\_ Method: Manages formal, developer-oriented string representation for debugging. Used by repr() and often displayed in interactive environments.

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

\_\_getitem\_\_: Intercepts slice operations and index-based retrieval. Handles index which can be either a slice or integer.

\_\_setitem\_\_: (Optional) Intercepts slice operations and index-based assignment. Handles setting values to a slice or index.

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

To capture in-place addition in a class, you implement the \_\_iadd\_\_ method. This method is responsible for handling the += operator, which modifies the object in place.

Q5. When is it appropriate to use operator overloading?

Appropriate Use: Mathematical classes, custom data structures, enhancing readability.

Guidelines: Be consistent with operator semantics, avoid misleading overloading, focus on essential methods, consider performance, and handle errors properly.