# Python OOP Questions & Answers (Set 4 - Operator Overloading)

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

Two ways:  
1. \_\_iter\_\_ with \_\_next\_\_ (iterator protocol).  
2. \_\_getitem\_\_ with successive indexes until IndexError (sequence protocol).

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

\_\_str\_\_: user-friendly string, used by print(), str(), f-strings.  
\_\_repr\_\_: developer/debug string, used by repr(), REPL, inside containers. Fallback if \_\_str\_\_ not defined.

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

Implement \_\_getitem\_\_, \_\_setitem\_\_, and \_\_delitem\_\_.  
Inside, check if key is a slice instance, then handle accordingly.

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

Implement \_\_iadd\_\_(self, other) for +=.  
Mutable types typically mutate self and return it.  
If missing, Python falls back to \_\_add\_\_ and rebinding.

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

Use when it makes code more natural and aligns with semantics:  
- Math/structured types (vectors, matrices, complex numbers).  
- Container-like behavior (iteration, indexing, slicing).  
- String/debug views (\_\_repr\_\_, \_\_str\_\_).  
Avoid when it surprises users or makes code less readable.