Python Classes & OOP Basics (Extended) — Q&A; (Set 19)

Q1. Relationship between a class and its instances

One-to-many: one class definition can produce many instances. Each instance is unique but follows the same class blueprint.

Q2. What kind of data is held only in an instance?

Instance attributes (self.x) — unique to each object, representing its state.

Q3. What kind of knowledge is stored in a class?

Class attributes and methods — shared by all instances, representing common data, defaults, or behavior.

Q4. What exactly is a method vs a regular function?

A method is a function defined in a class that takes self (or cls) as its first argument. A regular function is standalone and has no automatic instance context.

Q5. Is inheritance supported in Python? Syntax?

Yes. Use parentheses:
class Parent: ...
class Child(Parent): ...

Supports single and multiple inheritance.

Q6. How much encapsulation (privacy) does Python support?

Python relies on naming conventions, not strict enforcement:
- _name → soft private.
- __name → name mangling (_Class__name).
No true access restrictions.

Q7. Distinguishing class vs instance variables

Class variable: defined in class body, shared across all objects.
Instance variable: defined with self in methods, unique to each instance.

Q8. When is self included in method definitions?

Always for instance methods — represents the calling object.
Not needed in staticmethods.
In classmethods, cls replaces self.

Q9. Difference between __add__ and __radd__

__add__(self, other): handles self + other.
__radd__(self, other): handles other + self if other doesn’t support it.

Q10. When is it necessary to use reflection methods?

When implementing built-in operations like len(x) (calls x.__len__). Prefer built-ins (len, iter) when using them externally.

Q11. What is __iadd__ called?

In-place addition method, triggered by +=. Falls back to __add__ if not defined.

Q12. Is __init__ inherited by subclasses?

Yes. If subclass defines its own __init__, it overrides parent’s. To extend parent’s init, call super().__init__(...) inside the subclass.