Modules, __all__, __name__, and RPN Interpreter — Q&A; (Set 24)

Q1. Multiple imports of same module

Yes, allowed. Python loads a module once and caches it in sys.modules. Later imports return the cached object. Useful for conditional imports inside functions, importing under different aliases, or mocking in tests.

Q2. Characteristics of a module

A module is a namespace object with attributes like __name__, __file__, __doc__, __package__. It is a single live object stored in sys.modules.

Q3. Avoiding circular imports

Refactor common code into a third module. Move imports inside functions or use TYPE_CHECKING for hints. Use dependency injection to invert dependencies. Minimize top-level executable code.

Q4. Why __all__

__all__ defines the public API for from module import * by listing exported names. It does not restrict access via module.name.

Q5. When is __name__ or '__main__' useful

To distinguish run-as-script vs imported:
if __name__ == '__main__': main().
This lets a file serve as both a script and a library.

Q6. Benefits of program counter in RPN interpreter

Supports sequential execution and control flow (loops, jumps). Provides error line numbers, debugging (breakpoints), pause/resume, and deterministic state (PC + stack + memory).

Q7. Minimal primitives for complete RPN language

Needs sequencing, conditionals/loops, and memory.
- Stack ops: PUSH, DUP, SWAP, POP
- Arithmetic: ADD, SUB, MUL, DIV, CMPZ
- Memory: LOAD, STORE
- Control flow: JMP, JZ, HALT

This is sufficient for Turing completeness with unbounded memory.