Here's a concise "must-know" checklist of Python concepts that every professional developer is expected to have at their fingertips. Treat it as a syllabus: if you can write or explain a 5-line example for every bullet, you're interview-ready.

1. Built-ins & Syntax

- Indentation, comments, line continuations
- Variables, object identity (`id`), reference counting
- Truthy/falsy values, `is` vs `==`, short-circuit (`and/or`)
- f-strings, raw strings, triple quotes, `repr` vs `str`

2. Primitive Types

- `int`, `float`, `complex`, `Decimal` pitfalls
- `bytes`, `bytearray`, `memoryview`
- `str` methods, Unicode normalization, `encode`/`decode`

3. Containers

- `list` vs `tuple` (mutability, hashability, packing/unpacking)
- `set`/`frozenset`, set algebra, O(1) membership
- `dict` insertion order, views, `setdefault`, `collections.ChainMap`
- slicing (`[:]` copies), shallow vs deep (`copy` module)

4. Comprehensions & Generators

- list/dict/set comprehensions, nested loops, conditional clauses
- generator expressions, `yield`, `yield from`, `next(g, default)`
- `itertools` recipes: `islice`, `groupby`, `accumulate`, `product`

5. Functions

- positional, keyword, `*args`, `**kwargs`, keyword-only, `/` & `*`
- closures, late-binding trap in default args, `nonlocal`
- decorators (function → function), `@wraps`, parameterized decorators
- `lambda`, `operator` module, `functools.partial`, `singledispatch`

6. OOP

- class statement, `self`, `__init__`, instance vs class vs static methods
- dunder protocol: `__str__`, `__repr__`, `__len__`, `__bool__`, `__call__`
- attribute lookup order: instance → class → MRO → `__getattr__`
- inheritance, multiple inheritance, `super()` and MRO (C3 linearization)
- dataclasses (`@dataclass`), `__slots__`, property descriptors

7. Modules & Packages

- `import` vs `from`, `__all__`, relative imports, `PYTHONPATH`
- `if __name__ == "__main__"`, `python -m pkg.mod`
- namespace packages, `py.typed`, `__init__.py` optional in 3.3+

8. Virtual Environments & Packaging

- `venv`, `pip`, `requirements.txt`, `pip-tools`, `pipx`
- `pyproject.toml`, `setuptools`, `wheel`, `build`
- editable installs (`-e`), version specifiers, semantic versioning

9. Exceptions & Context

- exception hierarchy, catching specific errors, `else`, `finally`
- custom exceptions, exception chaining (`raise ... from`)
- context managers: `with`, `__enter__`/`__exit__`, `contextlib` utilities

10. Iteration & Async

- iterator protocol (`__iter__`, `__next__`), `StopIteration`
- `for...else`, `enumerate`, `zip`, `reversed`, `sorted` key functions
- `async`/`await`, event loop, `asyncio.gather`, `async for`, `async with`

11. Standard Library “Batteries”

- `pathlib` over `os.path`, `shutil`, `glob`, `tempfile`
- `datetime`, `zoneinfo`, `decimal`, `fractions`, `statistics`
- `argparse`, `logging` (handlers, formatters, filters), `configparser`
- `json`, `csv`, `sqlite3`, `subprocess`, `concurrent.futures`

12. Type Hints & Static Analysis

- `typing` module: `List`, `Dict`, `Optional`, `Union`, `Literal`, `TypedDict`
- generics (`TypeVar`, `Generic`), `Protocol`, `Final`, `cast`
- `mypy`, `pyright`, runtime checking with `typeguard`/`pydantic`

13. Testing & Debugging

- `unittest` vs `pytest` (fixtures, parametrize, markers)
- `assert` statements, `__debug__` flag, `-O`
- `pdb`, `breakpoint()`, post-mortem, `faulthandler`, `tracemalloc`

14. Performance & Memory

- big-O of list, dict, set operations
- `__slots__`, `array.array`, `numpy` when needed
- profiling: `cProfile`, `line_profiler`, `timeit`, `dis` bytecode

- GIL, multiprocessing vs threading, `concurrent.futures.ProcessPool`

15. Security & Hygiene

- never use `eval` on untrusted input, `ast.literal_eval`
- `secrets` over `random` for tokens, password hashing (`bcrypt`, `argon2`)
- dependency scanning (`pip-audit`, `safety`), virtualenv isolation

16. Style & Tooling

- PEP 8, PEP 257 (docstrings), `black`, `isort`, `flake8`, `pylint`
- pre-commit hooks, Git hooks, CI matrix (tox, nox)

17. Advanced (good-to-flaunt)

- metaclasses, descriptors (`__get__`, `__set__`, `__delete__`)
- `typing.Protocol` structural subtyping, runtime checkable
- pattern matching (`match`/`case`) Python 3.10+
- C extensions, `cffi`, `cython`, `mypyc`

Keep this list in a cheat-sheet; tick items off with tiny code katas. Once every bullet has a muscle-memory example, you've achieved "Pythonic fluency."