100 Python Interview Questions and Answers (Simple English)

Prepared for study. Each question has answer, explanation, and example code.

1. What is Python and why use it?

Answer: Python is a high-level, interpreted programming language used for many tasks like web, data, scripting, and automation.

Explanation (simple): Python is easy to read, has many libraries, and is good for beginners and professionals.

Example:
print('Hello, Python!')

2. How do you declare a variable in Python?

Answer: Just assign a value to a name, e.g., x = 5.

Explanation (simple): Python variables do not need a type declaration; types are inferred at runtime.

Example:
x = 5 name = 'Vijay' pi = 3.14

3. What are basic data types in Python?

Answer: int, float, str, bool, list, tuple, set, dict.

Explanation (simple): These are built-in types to store numbers, text, collections, and truth values.

Example:
a = 10 b = 2.5 s = 'hello' flag = True lst = [1,2,3]

4. What is the difference between list and tuple?

Answer: Lists are mutable (changeable), tuples are immutable (cannot change).

Explanation (simple): Use tuple for fixed data and list when you need to modify elements.

Example:
lst = [1,2] tpl = (1,2) lst[0] = 10 # OK # tpl[0] = 10 # Error

5. How do you write a function in Python?

Answer: Use def keyword: def func_name(params): ...

Explanation (simple): Functions group code for reuse. They can return values with return.

Example:
def add(a, b): return a + b print(add(2,3)) # 5

6. What is a for loop and while loop?

Answer: for loops iterate over sequences; while loops run while a condition is true.

Explanation (simple): Use for when you know the items; use while for conditions or unknown counts.

Example:
for i in range(3): print(i) n=0 while n<3: print(n) n+=1

7. How to handle exceptions in Python?

Answer: Use try, except, finally blocks.

Explanation (simple): Exceptions are errors; catch them to avoid program crash and handle errors cleanly.

Example:
try: x = 1/0 except ZeroDivisionError: print('Cannot divide by zero') finally: print('Done')

8. What are list comprehensions?

Answer: A concise way to create lists from iterables in one line.

Explanation (simple): They are shorter and often faster than loops for creating lists.

Example:
squares = [x*x for x in range(5)] # [0,1,4,9,16]

9. How do you read a file in Python?

Answer: Use open() with a context manager: with open('file') as f: ...

Explanation (simple): Context managers close the file automatically and avoid resource leaks.

Example:
with open('data.txt','r') as f: text = f.read() print(text)

10. What is a dictionary in Python?

Answer: A collection of key:value pairs (hash map).

Explanation (simple): Fast lookup by key; keys must be immutable and unique.

Example:
d = {'name':'Vijay', 'age':30} print(d['name']) # Vijay

11. How to add or remove items from a list?

Answer: Use append(), extend(), insert(), remove(), pop(), clear().

Explanation (simple): append adds single item; extend adds items from another list; pop removes by index.

Example:

lst = [1,2] lst.append(3) # [1,2,3] lst.remove(2) # [1,3] val = lst.pop() # 3

12. What is slicing?

Answer: Getting a part of sequence: seq[start:stop:step].

Explanation (simple): Works for list, tuple, string. Omitting start/stop uses defaults.

Example:
s = 'abcdef' print(s[1:4]) # 'bcd' print(s[::-1]) # reverse 'fedcba'

13. What are modules and packages?

Answer: Modules are .py files; packages are folders with __init__.py that group modules.

Explanation (simple): Use import to reuse code from other files and libraries.

Example:
import math from os import path import numpy as np # package example

14. What is the difference between import X and from X import Y?

Answer: import X loads module; from X import Y loads specific names into current namespace.

Explanation (simple): import keeps namespaced (X.Y). from-import allows direct use (Y).

Example:
import math print(math.sqrt(4)) from math import sqrt print(sqrt(4))

15. What is PEP 8?

Answer: PEP 8 is Python's style guide for clean readable code.

Explanation (simple): It recommends naming, spacing, and formatting rules to make code consistent.

Example:
Use meaningful_names, 4-space indent, max line 79 characters.

16. How to install external libraries?

Answer: Use pip: pip install package_name.

Explanation (simple): pip fetches packages from PyPI so you can use third-party tools.

Example:
pip install requests # then in code: import requests

17. What is virtual environment and why use it?

Answer: Isolated environment for Python projects to manage dependencies separately.

Explanation (simple): Prevents version conflicts between projects and keeps system clean.

Example:
python -m venv venv source venv/bin/activate # (Linux/Mac) venv\Scripts\activate # (Windows)

18. How do you create classes in Python?

Answer: Use class keyword and define methods with def; __init__ is constructor.

Explanation (simple): OOP groups data and behavior; classes create objects (instances).

Example:
class Person: def __init__(self, name): self.name = name def greet(self): return f'Hello {self.name}' p = Person('Vijay') print(p.greet())

19. What is inheritance?

Answer: A class can inherit attributes and methods from another class.

Explanation (simple): It helps reuse code and model 'is-a' relationships.

Example:
class Animal: def speak(self): return 'sound' class Dog(Animal): pass d = Dog() print(d.speak()) # sound

20. What are class and instance variables?

Answer: Class variables are shared by all instances; instance variables belong to each object.

Explanation (simple): Use class variables for shared defaults; instance variables for unique data.

Example:
class A: shared = 0 # class var def __init__(self, x): self.x = x # instance var

21. What is __init__ method?

Answer: Constructor that runs when creating an object.

Explanation (simple): It initializes instance variables and sets initial state.

Example:
class C: def __init__(self, v): self.v = v c = C(5) print(c.v) # 5

22. What is self in class methods?

Answer: Self refers to the instance calling the method.

Explanation (simple): It is the first argument of instance methods to access attributes.

Example:
class C: def setv(self, v): self.v = v c = C(); c.setv(10)

23. What are magic (dunder) methods?

Answer: Special methods like __str__, __repr__, __len__ that customize behavior.

Explanation (simple): They let objects work with built-in functions and operators.

Example:
class C: def __len__(self): return 5 print(len(C()))

24. What is iterator and iterable?

Answer: Iterable is an object you can loop over; iterator gives values one by one via __next__().

Explanation (simple): for loops call iter() then next() internally. Generators are a type of iterator.

Example:
it = iter([1,2]) print(next(it)) # 1 print(next(it)) # 2

25. What are generators?

Answer: Functions using yield to produce values lazily (one at a time).

Explanation (simple): They save memory for large sequences and support streaming data.

Example:
def gen(n): for i in range(n): yield i for x in gen(3): print(x)

26. What is lambda function?

Answer: Anonymous small functions written with lambda keyword.

Explanation (simple): Used when a short function is needed for a short time (e.g., sort key).

Example:
f = lambda x: x*2 print(f(3)) # 6 sorted([3,1,2], key=lambda x: -x)

27. What are map, filter, reduce?

Answer: map applies a function to each item, filter selects items by condition, reduce folds items to one value.

Explanation (simple): They are functional tools; reduce is in functools.

Example:
list(map(lambda x: x+1, [1,2])) list(filter(lambda x: x%2, [1,2,3])) from functools import reduce
reduce(lambda a,b: a+b, [1,2,3])

28. What is the difference between == and is?

Answer: == compares values; is checks object identity (same object in memory).

Explanation (simple): Use == to compare content, is rarely used (None check ok: x is None).

Example:
a = [1] b = [1] print(a == b) # True print(a is b) # False

29. How to check type of a variable?

Answer: Use type(x) or isinstance(x, Type).

Explanation (simple): isinstance is preferred for inheritance checks.

Example:
print(type(5)) print(isinstance(5, int))

30. What is duck typing?

Answer: If an object behaves like needed (has methods/attributes), it is used regardless of its class.

Explanation (simple): Python focuses on behavior, not strict types.

Example:
def quack(duck): duck.quack() # any object with quack() will work.

31. What is module __main__?

Answer: When running a script directly, its __name__ is '__main__'.

Explanation (simple): Use if __name__ == '__main__' to run code only when script is executed, not imported.

Example:
if __name__ == '__main__': main()

32. How to document code in Python?

Answer: Use docstrings (triple quotes) under functions, classes, and modules.

Explanation (simple): Docstrings help others and tools to see usage and behavior.

Example:
def add(a,b): '''Return sum of a and b''' return a+b

33. What are f-strings?

Answer: String formatting using f'...' introduced in Python 3.6.

Explanation (simple): They are concise and readable for embedding expressions.

Example:
name = 'Vijay' print(f'Hello {name}')

34. How to format strings with format()?

Answer: Use '{}'.format(value) or numbered placeholders.

Explanation (simple): Works on older Python and supports alignment and formatting.

Example:
print('Hi {}'.format('Vijay')) print('{:.2f}'.format(3.14159)) # 3.14

35. What are set and its typical uses?

Answer: An unordered collection of unique items used for membership tests and set operations.

Explanation (simple): Useful to remove duplicates and perform unions/intersections.

Example:
s = {1,2,2} print(s) # {1,2} print(1 in s) # True

36. How to merge dictionaries?

Answer: Use {**d1, **d2} or d1.update(d2) or in 3.9 d1 | d2.

Explanation (simple): Merging combines keys; later keys override earlier ones.

Example:
d1 = {'a':1} d2 = {'b':2} merged = {**d1, **d2}

37. What is JSON and how to use it in Python?

Answer: JSON is a text format for data; use json module to load/dump.

Explanation (simple): json.load/read to parse and json.dump/write to serialize.

Example:
import json s = json.dumps({'a':1}) print(json.loads(s))

38. How to make HTTP requests in Python?

Answer: Use requests library: requests.get/post etc.

Explanation (simple): It simplifies calling web APIs and handling responses.

Example:
import requests r = requests.get('https://httpbin.org/get') print(r.status_code) print(r.json())

39. What is threading and multiprocessing?

Answer: Threading uses threads (shared memory) for I/O-bound tasks; multiprocessing uses processes for CPU-bound tasks.

Explanation (simple): Use threading for tasks waiting on I/O; multiprocessing to use multiple CPU cores.

Example:
import threading import multiprocessing # simple examples omitted for brevity

40. How to debug Python code?

Answer: Use print statements, logging module, or pdb debugger and IDE tools.

Explanation (simple): logging is better for real apps; pdb allows stepping through code.

Example:
import pdb pdb.set_trace() # then run and inspect variables

41. What is the GIL (Global Interpreter Lock)?

Answer: A mutex in CPython that allows only one thread to execute Python bytecode at a time.

Explanation (simple): It limits CPU parallelism in threads; multiprocessing avoids GIL by using processes.

Example:
Use multiprocessing for CPU-heavy tasks; GIL affects threading performance.

42. What are decorators?

Answer: Functions that modify other functions by wrapping them.

Explanation (simple): They add behavior (like logging, timing) without changing function code.

Example:
def deco(f): def wrapper(*args, **kw): print('Before') res = f(*args, **kw) print('After') return res return wrapper @deco def hi(): print('Hello') hi()

43. What is closure?

Answer: A function that remembers values from its enclosing scope even when called later.

Explanation (simple): Closures help create functions with preserved state.

Example:
def outer(x): def inner(y): return x + y return inner f = outer(5) print(f(3)) # 8

44. What is property decorator?

Answer: Use @property to access methods like attributes for read-only or computed values.

Explanation (simple): It provides a clean API for classes without exposing method calls.

Example:
class C: def __init__(self,x): self._x = x @property def x(self): return self._x c=C(5) print(c.x)

45. How to work with dates and times?

Answer: Use datetime module for date/time objects; use pytz or zoneinfo for timezones.

Explanation (simple): datetime supports parsing, formatting, and arithmetic.

Example:
from datetime import datetime now = datetime.now() print(now.strftime('%Y-%m-%d %H:%M'))

46. What is unit testing in Python?

Answer: Writing tests for small parts of code using unittest or pytest.

Explanation (simple): Helps catch bugs early and ensures code works as expected.

Example:
import unittest class TestSum(unittest.TestCase): def test_add(self): self.assertEqual(1+1,2) if __name__=='__main__': unittest.main()

47. How to install a specific Python version?

Answer: Use pyenv (Linux/Mac) or installers from python.org or Windows store.

Explanation (simple): Different projects may need different Python versions; pyenv helps manage them.

Example:
pyenv install 3.10.6 pyenv global 3.10.6

48. What is pip freeze?

Answer: Lists installed packages with versions. Good to create requirements.txt.

Explanation (simple): Use pip freeze > requirements.txt to record dependencies for a project.

Example:
pip freeze > requirements.txt

49. What is requirements.txt?

Answer: A file listing a project's Python dependencies for reproducible installs.

Explanation (simple): Share it so others can install same packages with pip install -r requirements.txt.

Example:
pip install -r requirements.txt

50. What is async and await?

Answer: Keywords for asynchronous programming; async defines coroutine and await waits for it.

Explanation (simple): Useful for handling many I/O tasks without many threads.

Example:
import asyncio async def main(): await asyncio.sleep(1) print('done') asyncio.run(main())

51. Difference between synchronous and asynchronous code?

Answer: Synchronous runs step-by-step; asynchronous can start tasks and continue without waiting.

Explanation (simple): Async helps handle many tasks (like web requests) efficiently.

Example:
See async example above.

52. What is a byte and str difference?

Answer: str is text (Unicode); bytes is raw byte data. Encode/decode to convert.

Explanation (simple): Use bytes for binary files and network data; use str for readable text.

Example:
b = 'hi'.encode('utf-8') print(b) print(b.decode('utf-8'))

53. How to work with CSV files?

Answer: Use csv module or pandas for complex data; csv.reader and csv.writer for simple tasks.

Explanation (simple): CSV is a common text format for table data.

Example:
import csv with open('a.csv') as f: for row in csv.reader(f): print(row)

54. What is pandas and when to use it?

Answer: A library for data analysis providing DataFrame for table-like data.

Explanation (simple): Use pandas for cleaning, analyzing, and transforming datasets quickly.

Example:
import pandas as pd df = pd.DataFrame({'a':[1,2]}) print(df)

55. What is NumPy used for?

Answer: Library for fast numerical arrays and math operations.

Explanation (simple): It is more efficient than lists for numbers and used in scientific computing.

Example:
import numpy as np arr = np.array([1,2,3]) print(arr.mean())

56. How to sort a list of dictionaries by a key?

Answer: Use sorted with key=lambda x: x['key'].

Explanation (simple): sorted returns a new list; list.sort modifies in place.

Example:
people = [{'age':30},{'age':20}] sorted_people = sorted(people, key=lambda p: p['age'])

57. What is slicing negative indices?

Answer: Negative indices count from the end: -1 is last element.

Explanation (simple): Helps get tail elements easily.

Example:
lst = [1,2,3] print(lst[-1]) # 3 print(lst[-2:]) # [2,3]

58. How to copy lists and dictionaries?

Answer: Use copy() for shallow copy and copy.deepcopy() for deep copy.

Explanation (simple): Shallow copy copies top-level; nested objects still reference originals.

Example:
import copy l2 = l.copy() d2 = copy.deepcopy(d)

59. What is JSON serialization of custom objects?

Answer: Convert objects to dict or provide custom encoder to json.dumps.

Explanation (simple): json cannot directly dump custom objects; provide serializable form.

Example:
class C: def __init__(self,x): self.x=x obj = C(5) # json.dumps(obj.__dict__)

60. How to count occurrences in a list?

Answer: Use list.count(value) or collections.Counter for many counts.

Explanation (simple): Counter gives counts for all items efficiently.

Example:
from collections import Counter Counter([1,2,1]) # Counter({1:2,2:1})

61. What is enumerate()?

Answer: Gives index and value when looping over iterable.

Explanation (simple): Avoid manual index counters; clearer code.

Example:
for i, v in enumerate(['a','b']): print(i, v)

62. How to merge two lists element-wise?

Answer: Use zip() to pair elements together.

Explanation (simple): zip stops at shortest; use itertools.zip_longest for longer handling.

Example:
a=[1,2] b=[3,4] print(list(zip(a,b))) # [(1,3),(2,4)]

63. What is itertools?

Answer: A module with tools for efficient looping and combinatorics.

Explanation (simple): Includes product, permutations, combinations, accumulate, etc.

Example:
import itertools list(itertools.product([1,2],[3,4]))

64. How to measure time taken by code?

Answer: Use time.time(), time.perf_counter(), or timeit module for accurate timing.

Explanation (simple): perf_counter is best for timing short code segments.

Example:
import time start = time.perf_counter() # code end = time.perf_counter() print(end-start)

65. What are assertions?

Answer: Use assert to check conditions during development; raises AssertionError if false.

Explanation (simple): They are for debugging and can be disabled in production with -O flag.

Example:
assert 2+2==4 assert len([]) != 0 # will raise

66. What is logging module?

Answer: Flexible system to record messages with levels (DEBUG, INFO, WARNING, ERROR).

Explanation (simple): Better than print for production because logs can be saved and formatted.

Example:
import logging logging.basicConfig(level=logging.INFO) logging.info('Hello')

67. How to remove duplicates while keeping order?

Answer: Use dict.fromkeys(list) in Python 3.7+ or OrderedDict before 3.7.

Explanation (simple): dict keeps insertion order so this trick filters duplicates.

Example:
lst = [1,2,1] uniq = list(dict.fromkeys(lst)) # [1,2]

68. What is slicing with step?

Answer: seq[start:stop:step] where step skips elements by step size.

Explanation (simple): Useful for taking every n-th item or reversing with -1.

Example:
print(list(range(10))[::2]) # even indices

69. How to validate input from users?

Answer: Check types, ranges, and use try/except to handle bad input.

Explanation (simple): Never trust user input; validate to avoid errors and security issues.

Example:
try: n = int(input('Enter number:')) except ValueError: print('Not a number')

70. What is the difference between remove() and pop() for list?

Answer: remove(value) removes first matching value; pop(index) removes by index and returns it.

Explanation (simple): Use pop when you need removed value; remove when you know the value.

Example:
lst = [1,2,3] lst.remove(2) val = lst.pop(0) # returns 1

71. How to reverse a list?

Answer: Use reversed() iterator or list.reverse() method to reverse in place or slicing.

Explanation (simple): reversed returns iterator; list.reverse changes list.

Example:
lst = [1,2,3] print(list(reversed(lst))) lst.reverse() # now [3,2,1]

72. How to check if key exists in dict?

Answer: Use 'in' operator: if key in d: ...

Explanation (simple): Safe and readable way to test presence before accessing.

Example:
d = {'a':1} if 'a' in d: print(d['a'])

73. What is defaultdict?

Answer: collections.defaultdict creates dict with default factory for missing keys.

Explanation (simple): Avoid KeyError and manual checks when counting or grouping.

Example:
from collections import defaultdict d = defaultdict(int) d['a'] += 1 # 1

74. How to handle large files efficiently?

Answer: Read files line by line, use streaming, generators, and avoid loading all into memory.

Explanation (simple): Use with open(...) and iterate over file object.

Example:
with open('big.txt') as f: for line in f: process(line)

75. What is memory management in Python?

Answer: Python uses automatic memory management and garbage collection for unused objects.

Explanation (simple): Most of the time you don't manage memory manually; be careful with large references and cycles.

Example:
del obj # hint to free a reference

76. How to profile Python code?

Answer: Use cProfile or profile modules to see where time is spent.

Explanation (simple): Helps find bottlenecks for optimization.

Example:
import cProfile cProfile.run('main()')

77. What are context managers and how to make one?

Answer: Objects used with 'with' to set up and tear down resources; implement __enter__ and __exit__ or use contextlib.contextmanager.

Explanation (simple): They ensure correct cleanup even on errors.

Example:
from contextlib import contextmanager @contextmanager def cm(): print('setup') yield print('teardown') with cm(): print('inside')

78. How to use regex in Python?

Answer: Use re module with functions like re.search, re.match, re.findall.

Explanation (simple): Regex helps find patterns in text but can be complex; keep patterns clear.

Example:
import re m = re.search(r'\d+', 'ab12') print(m.group()) # '12'

79. What is serialization and pickle?

Answer: pickle serializes Python objects to bytes and back, for Python-specific persistence.

Explanation (simple): Not safe for untrusted data; prefer JSON for cross-language simple data.

Example:
import pickle b = pickle.dumps([1,2]) print(pickle.loads(b))

80. How to do command-line arguments parsing?

Answer: Use argparse module to parse flags and arguments with help and types.

Explanation (simple): It makes CLI programs user-friendly and robust.

Example:
import argparse p = argparse.ArgumentParser() p.add_argument('--num', type=int) args = p.parse_args()

81. What are namedtuples?

Answer: Like tuples but with named fields for readability.

Explanation (simple): They are immutable and lightweight records.

Example:
from collections import namedtuple P = namedtuple('P','x y') p = P(1,2) print(p.x)

82. What is typing and type hints?

Answer: Optional annotations for variables and functions to improve readability and tooling.

Explanation (simple): They don't change runtime but help static checkers like mypy.

Example:
def add(a: int, b: int) -> int: return a+b

83. How to handle CSV with pandas?

Answer: Use pd.read_csv and pd.to_csv for easy loading and saving.

Explanation (simple): Pandas handles headers, types, missing values and large data efficiently.

Example:
import pandas as pd df = pd.read_csv('data.csv') df.to_csv('out.csv', index=False)

84. What is a process vs thread?

Answer: Process is independent program with its own memory; thread is a path of execution inside a process sharing memory.

Explanation (simple): Processes are heavier but avoid GIL issues; threads lighter but GIL limits CPU parallelism in CPython.

Example:
Use multiprocessing.Process or threading.Thread

85. How to secure Python code (basic)?

Answer: Validate inputs, avoid eval on untrusted data, manage secrets, use latest libraries.

Explanation (simple): Security depends on context; follow best practices and use linters and tests.

Example:
DON'T: eval(user_input) DO: parse and validate input

86. How to connect to a database in Python?

Answer: Use database drivers (e.g., psycopg2 for Postgres, sqlite3 built-in) or ORM like SQLAlchemy.

Explanation (simple): ORM helps map Python objects to database rows; direct drivers give more control.

Example:
import sqlite3 conn = sqlite3.connect(':memory:') cur = conn.cursor() cur.execute('CREATE TABLE t(x)')

87. What is SQL injection and how to prevent it?

Answer: Injection is when untrusted input changes SQL commands; prevent with parameterized queries.

Explanation (simple): Never build SQL strings with user input directly.

Example:
cur.execute('SELECT * FROM users WHERE id=?', (user_id,)) # safe

88. What is REST API and how to build one?

Answer: REST is an architecture for web services using HTTP methods. Use frameworks like Flask or FastAPI.

Explanation (simple): Design endpoints for resources and use JSON for data exchange.

Example:
from flask import Flask, jsonify app = Flask(__name__) @app.route('/ping') def ping(): return jsonify({'pong':True})

89. What is FastAPI?

Answer: A modern, fast web framework for building APIs with automatic docs and async support.

Explanation (simple): It uses type hints to validate and document endpoints automatically.

Example:
from fastapi import FastAPI app = FastAPI() @app.get('/') def read_root(): return {'hello':'world'}

90. How to paginate large query results?

Answer: Use LIMIT/OFFSET or cursor-based pagination depending on needs; avoid loading everything into memory.

Explanation (simple): Cursor-based pagination is better for large shifting datasets.

Example:
SELECT * FROM table LIMIT 10 OFFSET 20

91. What are memory leaks in Python?

Answer: When references persist (e.g., global lists), objects never get freed leading to growing memory.

Explanation (simple): Use profiling and ensure references are removed; watch C-extensions too.

Example:

Avoid storing growing caches without limits.

92. How to write clean code in Python?

Answer: Follow PEP8, write small functions, meaningful names, docstrings, and tests.

Explanation (simple): Clean code is easier to maintain and debug.

Example:
Refactor duplicated code into functions and add comments where needed.

93. What is monkey patching?

Answer: Changing or extending code at runtime by assigning new functions or attributes.

Explanation (simple): Useful for quick fixes or tests but can make code hard to understand; use carefully.

Example:
import module module.func = lambda: 'patched'

94. How to work with images in Python?

Answer: Use PIL/Pillow library for opening, editing, and saving images.

Explanation (simple): Pillow is simple and good for common image tasks.

Example:
from PIL import Image img = Image.open('img.jpg') img.resize((100,100)).save('out.jpg')

95. What is Web scraping and which libraries to use?

Answer: Extracting data from websites using requests and BeautifulSoup or use Selenium for dynamic pages.

Explanation (simple): Respect robots.txt and site terms; avoid overloading servers.

Example:
import requests from bs4 import BeautifulSoup r = requests.get('https://example.com') soup = BeautifulSoup(r.text, 'html.parser') print(soup.title.string)

96. How to perform string encoding issues handling?

Answer: Always use UTF-8: open files with encoding='utf-8' and decode/encode bytes carefully.

Explanation (simple): Different systems may use different encodings; explicitly setting avoids errors.

Example:
with open('t.txt','r', encoding='utf-8') as f: print(f.read())

97. What is dependency injection?

Answer: Passing dependencies (like DB connection) into functions/classes instead of creating them

inside.

Explanation (simple): Improves testability and decouples components.

Example:
class Service: def __init__(self, repo): self.repo = repo # pass different repo in tests

98. How to implement caching in Python?

Answer: Use functools.lru_cache for function results or external caches like Redis.

Explanation (simple): Caching speeds repeated computations but needs invalidation strategy.

Example:
from functools import lru_cache @lru_cache(maxsize=128) def fib(n): if n<2: return n return fib(n-1)+fib(n-2)

99. What are common interview algorithms in Python to practice?

Answer: Sorting, searching, recursion, dynamic programming, two pointers, BFS/DFS, hash maps.

Explanation (simple): Practice coding patterns, time and space complexity, and example problems.

Example:
Example: two-sum def two_sum(nums, target): seen = {} for i, n in enumerate(nums): if target-n in seen: return [seen[target-n], i] seen[n] = i

100. How to prepare for Python interviews?

Answer: Practice coding problems, understand core concepts, build small projects, and explain solutions aloud.

Explanation (simple): Focus on clarity, correctness, and writing clean code with tests and examples.

Example:
Practice on sites like LeetCode, HackerRank, build a small project and review common patterns.