# Files and Exception Handling - Python Interview Q&A

1. 1. How do you read and write files in Python?

Answer: You can read a file using open('filename', 'r') and write using open('filename', 'w'). Use read(), readline(), or readlines() to read and write() to write.

1. 2. What are the differences between reading a file in text mode and binary mode?

Answer: Text mode reads data as strings, binary mode reads as bytes. Use 'r' for text and 'rb' for binary.

1. 3. What is the difference between the open() function's 'r', 'w', 'a', and 'x' modes?

Answer: 'r' = read, 'w' = write (overwrite), 'a' = append, 'x' = create new file. Example: open('file.txt', 'a') appends data.

1. 4. How do you read a large file line-by-line in Python efficiently?

Answer: Use a loop: with open('file.txt') as f: for line in f: process(line). This avoids loading the entire file into memory.

1. 5. What are file pointers in Python, and how can they be manipulated?

Answer: File pointers indicate current position in a file. Use f.tell() to get position, f.seek(pos) to move pointer.

1. 6. What is diamond problem in OOPS?

Answer: The diamond problem occurs in multiple inheritance when two parent classes have the same method name, causing ambiguity in the child class.

1. 7. Explain the use of reduce function?

Answer: reduce() applies a function cumulatively to all items in an iterable. Example: reduce(lambda x, y: x + y, [1,2,3,4]) gives 10.

1. 8. What is method overloading and overriding? Difference between them.

Answer: Overloading: same method name, different parameters (not directly supported in Python). Overriding: child class redefines a parent class method.

1. 9. What is inheritance? Different types of inheritance?

Answer: Inheritance allows a class to derive from another. Types: single, multiple, multilevel, hierarchical, hybrid.

1. 10. What is polymorphism? How it can be achieved?

Answer: Polymorphism allows one interface to be used for different types. Achieved using method overriding and duck typing.

1. 11. What is exception handling in Python, and why is it necessary?

Answer: Exception handling manages runtime errors to prevent program crash using try, except, finally blocks.

1. 12. How do you raise an exception in Python?

Answer: Use raise Exception('message') to manually raise an exception.

1. 13. What are try, except, and finally blocks in Python? Provide examples.

Answer: try: block for risky code, except: handle errors, finally: executes always. Example: try: x=1/0 except: print('error') finally: print('done')

1. 14. What is the difference between interpreted and compiled languages?

Answer: Compiled languages convert code to machine code before execution; interpreted languages run line by line. Python is interpreted.

1. 15. What is the purpose of the with statement when handling files?

Answer: The with statement handles file closing automatically. Example: with open('file.txt') as f: data = f.read()

1. 16. How do you handle multiple exceptions in a single try-except block?

Answer: Use multiple except blocks or a tuple: except (TypeError, ValueError): handle both.

1. 17. What is the importance of exception hierarchy in Python?

Answer: Exception hierarchy ensures specific exceptions are caught before general ones for better control.

1. 18. How does custom exception handling work in Python? How do you define your own exceptions?

Answer: Custom exceptions are user-defined by subclassing Exception. Example: class MyError(Exception): pass

1. 19. What is the difference between checked and unchecked exceptions? Does Python enforce them?

Answer: Checked exceptions are verified at compile time, unchecked at runtime. Python does not enforce checked exceptions.

1. 20. Explain the difference between an interpreted language and a compiled language. How does Python fit in?

Answer: Compiled languages translate code before running; interpreted run line-by-line. Python is interpreted.

1. 21. Why is Python considered slower compared to compiled languages like C++?

Answer: Python is slower because it’s interpreted and dynamically typed, unlike C++ which is compiled and optimized.

1. 22. Can you speed up Python execution? What strategies would you employ?

Answer: Use libraries like NumPy, Cython, or PyPy; optimize loops, use vectorization, and multiprocessing.

1. 23. How does Python's logging module work? Explain different logging levels (DEBUG, INFO, WARNING, ERROR).

Answer: logging module records events. Levels: DEBUG, INFO, WARNING, ERROR, CRITICAL.

1. 24. What is the difference between logging and debugging in Python?

Answer: Logging records events; debugging inspects and fixes errors during development.

1. 25. How would you implement a basic logging mechanism in your Python code?

Answer: Basic logging: import logging; logging.basicConfig(level=logging.INFO); logging.info('Message')

1. 26. Can you set logging configuration in Python using a configuration file?

Answer: Yes, by using logging.config.fileConfig('logging.conf') to set configuration from a file.

1. 27. What is the difference between a library, a package, and a module in Python?

Answer: Module = single file, Package = folder with \_\_init\_\_.py, Library = collection of packages.

1. 28. How does the import statement work in Python? How is it different from from module import?

Answer: import module imports whole module; from module import func imports specific part.

1. 29. What is the role of \_\_init\_\_.py in a Python package?

Answer: \_\_init\_\_.py marks a directory as a package and can run initialization code.