Python FAQ PDF 3: Advanced Python Topics & Best Practices

Q1: What is multithreading in Python and when should it be used?
A: Multithreading allows a Python program to run multiple threads (smaller units of a process) concurrently. Python's threading module provides this capability, but due to the Global Interpreter Lock (GIL), only one thread executes Python bytecode at a time. Multithreading is best used for I/O-bound tasks (like file or network operations), not CPU-bound tasks.

Q2: How does multiprocessing differ from multithreading in Python?
A: Multiprocessing runs separate processes, each with its own Python interpreter and memory space, allowing true parallel execution. This approach bypasses the GIL and is suitable for CPU-bound tasks. The multiprocessing module makes it easy to parallelize work across multiple CPU cores.

Q3: What is asynchronous programming in Python?
A: Asynchronous programming enables a program to handle many operations at once, such as I/O-bound tasks, without waiting for each to complete sequentially.
Python's asyncio library provides tools for asynchronous programming
using async and await keywords, allowing efficient handling of tasks like network requests.

Q4: What are context managers and how are they implemented?
A: Context managers allow for setup and cleanup actions to be performed automatically, typically using the with statement (e.g., opening files). A context manager implements __enter__ and __exit__ methods. Custom context managers can be created using classes or the contextlib module's contextmanager decorator.

Q5: How does Python support unit testing?
A: Python's unittest module provides a framework for writing and running tests. Tests are organized into test cases by subclassing unittest.TestCase, with methods that test specific functionality. Additional testing frameworks like pytest and nose offer more features and easier syntax.

Q6: What is the purpose of the __init__.py file in a Python package?
A: The __init__.py file indicates that a directory is a Python package. It can be empty or execute initialization code for the package. It allows for package imports and can define which modules are exposed when the package is imported.

Q7: How do you handle dependencies in a Python project?
A: Dependencies are typically listed in a requirements.txt file, which can be generated using pip freeze. Tools like pip and venv manage these dependencies, ensuring consistent environments across development and production.

Q8: What are Python decorators and how can they be chained?
A: Decorators are functions that modify the behavior of other functions or methods. They

can be chained by stacking multiple decorators above a function definition. Decorators are powerful tools for code reuse, logging, authentication, and more.

Q9: How does Python manage function arguments with *args and **kwargs?
A: The *args syntax allows a function to accept any number of positional arguments, while **kwargs allows for any number of keyword arguments. This flexibility enables functions to handle varying argument lists and pass arguments to other functions.

Q10: What are some best practices for writing Pythonic code?
A: Best practices include following PEP 8 for style, using meaningful variable names, leveraging built-in functions and libraries, writing modular and reusable code, handling exceptions gracefully, documenting code with docstrings, and writing unit tests. Writing "Pythonic" code means using idioms and patterns that are natural and efficient in Python.