Question 1:
FastAPI leverages asynchronous programming to handle HTTP requests efficiently. It utilizes asynchronous features provided by Python 3.7+ through the async and await keywords along with the asyncio library.

Here's how FastAPI handles asynchronous requests and its benefits over synchronous code in Python:

Asynchronous I/O Operations:
FastAPI can handle multiple I/O-bound tasks concurrently without blocking the execution flow. This is achieved using asynchronous functions and the await keyword, allowing the server to handle multiple requests simultaneously.

Performance:
Asynchronous code allows the server to handle a large number of concurrent connections with fewer resources. This results in better performance and scalability, making FastAPI suitable for high-throughput applications.

Improved Responsiveness:
Asynchronous handling ensures that the server remains responsive even under heavy loads. Requests can be processed in parallel, reducing overall latency and improving user experience.

Simplified Code:
FastAPI simplifies asynchronous programming by providing decorators and utilities to define asynchronous endpoints easily. Developers can write asynchronous code in a straightforward manner, making it easier to maintain and debug.

Compatibility:
FastAPI seamlessly integrates with asynchronous libraries and frameworks, allowing developers to leverage existing asynchronous code and libraries in their applications.

Overall, FastAPI's asynchronous support enables developers to build high-performance web APIs that can efficiently handle concurrent requests, resulting in better scalability and responsiveness.

Question 2:
Dependency injection in FastAPI allows you to inject dependencies into your endpoint functions automatically. This helps in managing and organizing code, promoting reusability, and simplifying testing.

Here's how dependency injection works in FastAPI:

Automatic Dependency Resolution:
FastAPI uses the Depends class to automatically resolve dependencies for endpoint functions. Dependencies can be defined as parameters in the endpoint function signature, and FastAPI takes care of instantiating and injecting them when the endpoint is called.

Dependency Injection Containers:

FastAPI provides a built-in dependency injection container that manages the lifecycle of dependencies and ensures they are instantiated only once per request. This helps in managing resources efficiently and avoiding unnecessary overhead.

Dependency Declaration:

Dependencies can be declared using Python type hints or by defining them as classes with appropriate initialization methods. FastAPI uses type annotations to determine the dependencies required by an endpoint function and resolves them accordingly.

Question 3:

```
from fastapi import Depends, FastAPI

app = FastAPI()

# Dependency
async def get_user_agent(user_agent: str = Depends(lambda x:
x.headers["user-agent"])):
    return {"user_agent": user_agent}

@app.get("/")
async def read_root(user_agent: str = Depends(get_user_agent)):
    return {"User-Agent": user_agent}
```