Async I/O in Python programming Language

This is a documentation for the following web-page :
https://sites.google.com/view/kolledge/advanced/async-io

Contents of the Web Page are as follows :

- **Event loop in Python :** The event loop is a central concept in Python's asynchronous programming paradigm, commonly used with the 'asyncio' module. It acts as a coordinator, managing the execution of asynchronous tasks and events. The event loop allows non-blocking I/O operations, enabling efficient utilization of resources and responsiveness in concurrent applications.
- **Async-await in Python :** Async-await is a syntax feature in Python that simplifies asynchronous programming by allowing developers to write asynchronous code that resembles synchronous code. By using the 'async' keyword with functions and 'await' keyword to pause execution, developers can create asynchronous tasks and make non-blocking calls, enhancing the readability and maintainability of asynchronous code.
- **Creating tasks in Python :** In Python's asynchronous programming, tasks represent units of concurrent execution that can be scheduled and executed by the event loop. Developers can create tasks using the 'asyncio.create_task()' function, allowing them to perform multiple asynchronous operations concurrently.
- **Canceling tasks in Python :** In Python's asynchronous programming, tasks can be canceled if they are no longer needed or have timed out. The 'asyncio.Task.cancel()' method is used to

cancel tasks, releasing resources and stopping their execution gracefully.

- **Asyncio.wait_for() function in Python :** The 'asyncio.wait_for()' function in Python's 'asyncio' module allows developers to set a maximum timeout for awaiting a coroutine. If the awaited coroutine does not complete within the specified timeout, a 'TimeoutError' is raised, enabling better control over asynchronous operations with a time limit.
- **Asyncio future object in Python :** In Python's 'asyncio' module, a future is an object that represents the result of a coroutine before it has completed. It provides a way to interact with the coroutine's outcome, allowing developers to track the status of asynchronous operations and handle their results appropriately.
- **Asyncio.gather() function in Python :** The 'asyncio.gather()' function in Python's 'asyncio' module allows developers to run multiple coroutines concurrently and gather their results. By specifying a list of coroutines to execute, developers can efficiently combine the results of multiple asynchronous operations into a single future object.