9 модуль. Задания

Глава 9.2. asyncio.Lock и блокировки

1. Что такое asyncio.Lock()?

   – Метод для создания многопоточных приложений.
   – Класс для управления синхронизацией, защищающий ресурсы от одновременного доступа.
   – Метод для создания циклов в Python.
   – Метод для создания асинхронных функций.

   Правильный ответ: Класс для управления синхронизацией, защищающий ресурсы от одновременного доступа.

2. Какой метод используется для запроса блокировки в asyncio.Lock()?

   – lock.request()
   – lock.acquire()
   – lock.get()
   – lock.lock()

   Правильный ответ: lock.acquire()

3. Какой метод используется для освобождения блокировки в asyncio.Lock()?

   – lock.free()
   – lock.release()
   – lock.remove()
   – lock.unlock()

   Правильный ответ: lock.release()

4. Что возвращает метод lock.locked() в asyncio.Lock()?

   – Количество блокировок, установленных в данный момент.
   – Идентификатор блокировки.
   – Время, когда блокировка была установлена.
   – True, если блокировка в данный момент захвачена, и False в противном случае.

   Правильный ответ: True, если блокировка в данный момент захвачена, и False в противном случае.

5. Что произойдет, если вы не освободите блокировку в asyncio.Lock()?

   – Блокировка будет переназначена другому потоку.
   – Блокировка автоматически освободится после определенного времени.
   – Это может привести к блокировке всего приложения.
   – Программа завершится с ошибкой.

   Правильный ответ: Это может привести к блокировке всего приложения.

6. Что делает менеджер контекста async with в контексте asyncio.Lock()?

   – Он автоматически обновляет блокировку после использования.
   – Он автоматически создает новую блокировку после использования.
   – Он автоматически освобождает блокировку после использования.
   – Он автоматически обрабатывает исключения, которые могут возникнуть при использовании блокировки.

   Правильный ответ: Он автоматически освобождает блокировку после использования.

7. Что происходит, когда несколько корутин пытаются одновременно получить доступ к общему ресурсу без использования asyncio.Lock()?

– Разные корутины могут перезаписывать друг друга, что приведет к непредсказуемому поведению программы.
– Программа завершится с ошибкой.
– Корутины будут выполняться последовательно. Корутины будут выполняться параллельно без каких-либо проблем.

Правильный ответ: Разные корутины могут перезаписывать друг друга, что приведет к непредсказуемому поведению программы.

Глава 9.3. asyncio.Event и события

1. Что такое asyncio.Event()?

– Это метод для обработки событий.
– Это класс, используемый для синхронизации между задачами в асинхронном коде.
– Это модуль для работы с событиями.
– Это функция для создания событий.

Правильный ответ: Это класс, используемый для синхронизации между задачами в асинхронном коде.

2. Для чего используется asyncio.Event()?

– Для управления последовательностью выполнения задач в асинхронном коде.
– Для создания событий в GUI приложениях.
– Для обработки событий ввода-вывода.
– Для создания многопоточных приложений.

Правильный ответ: Для управления последовательностью выполнения задач в асинхронном коде.

3. Что представляет собой объект asyncio.Event()?

– Список событий.
– Словарь событий.
– Булевый флаг, который может быть установлен в True или False.
– Строку события.

Правильный ответ: Булевый флаг, который может быть установлен в True или False.

4. Что делает метод event.set() в asyncio.Event()?

– Блокирует выполнение задачи, пока флаг события не будет установлен в True. Устанавливает флаг события в False.
– Устанавливает флаг события в True.
– Возвращает True, если флаг события установлен в True, и False в противном случае.

Правильный ответ: Блокирует выполнение задачи, пока флаг события не будет установлен в True. Устанавливает флаг события в False.

5. Что делает метод event.clear() в asyncio.Event()?

– Устанавливает флаг события в True.
– Возвращает True, если флаг события установлен в True, и False в противном случае. Блокирует выполнение задачи, пока флаг события не

будет установлен в True. Устанавливает флаг события в False.

Правильный ответ: Возвращает True, если флаг события установлен в True, и False в противном случае. Блокирует выполнение задачи, пока флаг события не будет установлен в True. Устанавливает флаг события в False.

6. Что делает метод event.wait() в asyncio.Event()?

– Устанавливает флаг события в False.
– Устанавливает флаг события в True.
– Блокирует выполнение задачи, пока флаг события не будет установлен в True.
– Возвращает True, если флаг события установлен в True, и False в противном случае.

Правильный ответ: Блокирует выполнение задачи, пока флаг события не будет установлен в True.

7. Что делает метод event.is_set() в asyncio.Event()?

– Устанавливает флаг события в False.
– Блокирует выполнение задачи, пока флаг события не будет установлен в True.
– Возвращает True, если флаг события установлен в True, и False в противном случае. Устанавливает флаг события в True.

Правильный ответ: Возвращает True, если флаг события установлен в True, и False в противном случае. Устанавливает флаг события в True.

Глава 9.4. asyncio.Condition и условия

1. Что такое asyncio.Condition()?

– Метод для создания многопоточных приложений.
– Метод для создания синхронных приложений.
– Это метод синхронизации потоков, который позволяет одному потоку ожидать условия, которое должен установить другой поток.
– Метод для создания асинхронных приложений.

Правильный ответ: Это метод синхронизации потоков, который позволяет одному потоку ожидать условия, которое должен установить другой поток.

2. Какой метод используется для захвата условия в asyncio.Condition()?

– Метод release()
– Метод wait()
– Метод acquire()
– Метод notify()

Правильный ответ: Метод acquire()

3. Что делает метод wait() в asyncio.Condition()?

– Освобождает условие.
– Блокирует текущую корутину, дожидаясь уведомления от другой корутины.
– Уведомляет другую корутину о готовности текущей корутины.
– Захватывает условие.

Правильный ответ: Блокирует текущую корутину, дожидаясь уведомления от другой корутины.

4. Что делает метод notify() в asyncio.Condition()?

– Уведомляет одну из корутин, которые ожидают условия.
– Блокирует текущую корутину, дожидаясь уведомления от другой корутины.
– Освобождает условие.
– Захватывает условие.

Правильный ответ: Уведомляет одну из корутин, которые ожидают условия.

5. Какой метод возвращает True, если условие сейчас захвачено, и False в противном случае?

– Метод wait()
– Метод locked()
– Метод notify()
– Метод acquire()

Правильный ответ: Метод wait()

6. Какой из следующих утверждений верен для asyncio.Condition()?

– asyncio.Condition() не позволяет использовать контекстные менеджеры.
– asyncio.Condition() сочетает в себе функциональность Event и Lock.
– asyncio.Condition() используется только для создания синхронных приложений. asyncio.Condition() не поддерживает многопоточность.

Правильный ответ: asyncio.Condition() сочетает в себе функциональность Event и Lock.

Глава 9.5. asyncio.Semaphore и семафоры

1. Что такое семафор в контексте asyncio в Python?

– Это примитив синхронизации, который позволяет ограничить количество потоков, которые могут захватить блокировку.
– Это инструмент для создания многопоточных приложений.
– Это функция для запуска асинхронных операций.
– Инструмент для управления временем выполнения потоков.

Правильный ответ: Это примитив синхронизации, который позволяет ограничить количество потоков, которые могут захватить блокировку.

2. Какой метод семафора запрашивает доступ к ресурсу?

– semaphore.acquire()
– semaphore.release()
– semaphore._value()
– semaphore.locked()

Правильный ответ: semaphore.acquire()

3. Что делает метод semaphore.locked()?

– Возвращает True, если семафор не может быть получен немедленно.
– Запрашивает доступ к ресурсу.
– Увеличивает счетчик на 1.
– Освобождает ресурс.

Правильный ответ: Возвращает True, если семафор не может быть получен немедленно.

4. Что происходит, когда счетчик семафора достигает нуля?

– Все доступные ресурсы используются, и новые задачи должны ждать, пока не освободится какой-либо ресурс.
– Семафор автоматически уменьшает счетчик на 1.
– Все задачи останавливаются.
– Семафор автоматически увеличивает счетчик на 1.
– Что делает следующая конструкция: async with semaphore:
– Освобождает семафор.
– Получает семафор и автоматически освобождает его после выполнения блока кода. Создает новый семафор.
– Проверяет, заблокирован ли семафор.

Правильный ответ: Все доступные ресурсы используются, и новые задачи должны ждать, пока не освободится какой-либо ресурс.

5. Какой метод используется для установки верхнего предела счетчика при создании экземпляра asyncio.Semaphore()?

– set_counter()
– Экземпляр asyncio.Semaphore создается с указанием предела в качестве аргумента.
– limit()
– set_limit()

Правильный ответ: set_counter()

Глава 9.6. asyncio.BoundedSemaphore и ограниченные семафоры

1. Что такое ограниченный семафор (BoundedSemaphore)?

– Это семафор, который может быть использован только в определенных условиях.
– Это семафор, который может быть использован только внутри определенного модуля.
– Это семафор, который может быть использован только внутри определенного класса.
– Это семафор, у которого верхний предел не может быть увеличен.

Правильный ответ: Это семафор, у которого верхний предел не может быть увеличен.

2. Как проверить, заблокирован ли BoundedSemaphore в Python?

– Используя метод check_lock()
– Используя метод locked()
– Используя метод status()
– Используя метод is_locked()

Правильный ответ: Используя метод locked()

3. Чем asyncio.BoundedSemaphore() отличается от asyncio.Semaphore()?

– asyncio.BoundedSemaphore() не позволяет внутреннему счетчику превышать начальное значение.
– asyncio.BoundedSemaphore() работает быстрее, чем asyncio.Semaphore().
– asyncio.BoundedSemaphore() использует меньше памяти, чем asyncio.Semaphore().
– asyncio.BoundedSemaphore() работает только с асинхронными функциями.

Правильный ответ: asyncio.BoundedSemaphore() не позволяет внутреннему счетчику превышать начальное значение.

4. В каком случае лучше использовать asyncio.BoundedSemaphore() вместо asyncio.Semaphore()?

– Когда нужно использовать семафор в многопоточной программе.
– Когда нужно гарантировать, что внутренний счетчик семафора не превысит начальное значение.
– Когда нужно сэкономить память.
– Когда нужно ускорить выполнение программы.

Правильный ответ: Когда нужно гарантировать, что внутренний счетчик семафора не превысит начальное значение.

Глава 9.7. asyncio.Barrier и барьеры

1. Что такое asyncio.Barrier() ?

– Это метод для создания синхронных задач.
– Это класс для синхронизации нескольких асинхронных задач в одном месте.
– Это функция для создания асинхронных задач.
– Это класс для создания многопоточных задач.

Правильный ответ: Это класс для синхронизации нескольких асинхронных задач в одном месте.

2. Какой метод используется для ожидания достижения барьера всеми задачами в asyncio.Barrier()?

– barrier.wait()
– barrier.hold()
– barrier.stop()
– barrier.pause()

Правильный ответ: barrier.wait()

3. Что произойдет, если количество задач, зарегистрированных на барьере, меньше, чем указано при создании объекта barrier?

– Программа выдаст ошибку.
– Барьер будет автоматически увеличен до необходимого числа.
– Программа завершится нормально.
– Метод wait() будет заблокирован навсегда.

Правильный ответ: Программа выдаст ошибку.

4. Что делает следующий код?
```
import asyncio
barrier = asyncio.Barrier(3)
```

– Создает барьер для трех синхронных задач.
– Создает три асинхронные задачи без барьера.
– Создает барьер для трех асинхронных задач.
– Создает три асинхронные задачи и ожидает их завершения.

Правильный ответ: Создает барьер для трех асинхронных задач.

5. Что делает следующий код?
```
await barrier.wait()
```

– Ожидает, пока все задачи не начнутся.

– Ожидает, пока все задачи не завершатся.
– Ожидает, пока все задачи не достигнут барьера.
– Ожидает, пока все задачи не будут отменены.

Правильный ответ: Ожидает, пока все задачи не начнутся.

6. Что происходит, когда все задачи достигают барьера в asyncio.Barrier()?

– Все задачи удаляются.
– Все задачи останавливаются.
– Все задачи начинаются заново.
– Все задачи разблокируются и могут продолжить свое выполнение.

Правильный ответ: Все задачи разблокируются и могут продолжить свое выполнение.