**Последовательность работы с очередями**

1. **Создание очереди**

Создайте объект очереди с использованием функции DispatchQueue(label:). Можно создавать как сериальные, так и параллельные очереди.

1. let serialQueue = DispatchQueue(label: "com.example.serialQueue")
2. let concurrentQueue = DispatchQueue(label: "com.example.concurrentQueue", attributes: .concurrent)
3. **Добавление задач в очередь**

Создайте блок кода, представляющий задачу, и добавьте его в очередь с использованием метода async или sync (для синхронного выполнения).

1. **Синхронное и асинхронное выполнение**

Решите, нужно ли выполнять задачу асинхронно или синхронно. В большинстве случаев рекомендуется использовать асинхронное выполнение для избежания блокировок.

1. serialQueue.async {
2. // Ваш код для выполнения в сериальной очереди
3. }
4. concurrentQueue.async {
5. // Ваш код для выполнения в параллельной очереди
6. }
7. **Использование глобальных очередей**

Для фоновых задач используйте глобальные очереди с разными приоритетами.

1. let globalQueue = DispatchQueue.global(qos: .background)
2. globalQueue.async {
3. // Ваш код для выполнения в фоновом потоке
4. }
5. **Ожидание выполнения группы задач**

Если у вас есть группа задач, используйте DispatchGroup, чтобы дождаться их завершения.

1. let group = DispatchGroup()
2. serialQueue.async(group: group) {
3. // Задача 1
4. }
5. serialQueue.async(group: group) {
6. // Задача 2
7. }
8. group.notify(queue: DispatchQueue.main) {
9. // Код, выполняемый после завершения всех задач
10. }
11. **Освобождение ресурсов и отмена задач**

Если необходимо освободить ресурсы или отменить выполнение задач, управляйте их жизненным циклом и используйте соответствующие методы, такие как cancel для отмены.