1. Typically, future is like a proxy which represents a result of thread. Not all the time you need a thread to complete your task or job.
2. Example, Sometimes you want to create a Future representing some event that you know will occur, e.g. [JMS message arrival](http://nurkiewicz.blogspot.no/2013/03/deferredresult-asynchronous-processing.html). So you have Future<Message> but there is no asynchronous job underlying this future.
3. You simply want to complete (resolve) that future when JMS message arrives, and this is driven by an event.
4. In this case you can simply create CompletableFuture, return it to your client and whenever you think your results are available, simply **complete()** the future and unlock all clients waiting on that future.

**public** CompletableFuture<String> localMethodCompletableFuture1(){

**final** CompletableFuture<String> future = **new** CompletableFuture<>();

future.complete(remoteMethod());

**return** future;

}

Sometimes you want to signal failure. As you know Future objects can handle either wrapped result or exception. If you want to pass some exception further, there isCompletableFuture.completeExceptionally(ex)

there is alsoCompletableFuture.join()

<http://allegro.tech/2014/10/async-rest.html>