Give an alternative to actor blocking.

Arrange for a message to arrive designating that an action is ready to be performed. This can be done by having a helper actor block for you.

```
actor {
  Thread.sleep(time)
  mainActor ! "WAKEUP"
}
```

Give the four points of good actors style.

- Actors should not block.
- They should communicate with actors only via messages.
- They should prefer immutable messages.
- They should make messages self-contained.

Why should messages be self-contained?

The caller may not know what state it was in when it made the original request since it did not block on the answer.

Why is react more efficient than receive?

Threads can more easily put actors in "cold storage" since react does not return and thus preserve the current thread's call stack.

When is it safe to send a mutable object as a method?

Why is it still a bad idea?

If the actor never reads or writes from the object after sending it.

It's a terrible idea because of future maintenance.

How do Scala actors receive messages?

Through a receive method which takes a Partial Function normally defined by a series of case statements.

How can you preserve threads when using Actors?

How is it different?

Use react instead of receive, which never returns.

Behind the scenes, react throws an exception just before falling off.

What are some good ways to make messages self-contained?

- If the request is immutable, return a copy of the request with the reply.
- More generally, wrap request and response info into case classes that include return-address in request, and request in the result.

What's Scala's approach to threading?

Share-nothing, message passing.

What's the key way the actors model addresses the difficulties of the shared data and locks model?

By providing a safe space - the actor's act method - where you can think sequentially.

Mutable local objects can be used inside act because each act method is effectively confined to one thread.

- so long as they aren't received or passed!

Give a basic Actor example.

```
object MyActor extends scala.actors.Actor
  def act() = println("concurrent!")
MyActor.start()
```

What do you do if you need to send a mutable object to another actor?

Send a copy instead.

What's Java's concurrency approach?

Shared data and locks.

What's the special library support for `react()`?

Actor.loop

```
def act() {
  loop {
    react {
      case...
      case...
     }
  }
}
```

Why shouldn't actors block?

- The actor will not notice requests it receives when it is blocked.
- Deadlock can result from multiple actors blocking waiting for other blocked actors to respond.

What's the best way to provide actors access to mutable data?

Make one actor that "owns" the data. It is the only one that accesses the data directly.

All other actors access the data by sending messages to the owner and waiting for a reply.

What should you do if the mailbox contains no messages?

Receive blocks until it receives a message for which `isDefinedAt` returns true. Others are silently ignored.

What's a quick way to make a new Actor?

How does it work?

`scala.actors.Actor.actor` factory method

Takes a block of code (=>Unit) to be executed by the newly created actor, which is automatically started.

Sending a message does not __.

Receiving a message is __.

block.

not interrupted.

Use `Thread.current` as an Actor.

Use `Actor.self`.

It's best to use `receiveWithin(millis)` to allow timeout. Otherwise you may block the interpreter shell.

Create an actor that echoes back the `Ints` it receives.

```
val echoActor = actor {
  while(true) {
    receive {
      case intMsg: Int =>
        println("received int" + intMsg)
    }
  }
}
```

What is useful about using `PartialFunctions` over allowing non-exhaustive `FunctionNs`?

`PartialFunction` provides `isDefaultAt()` to avoid runtime errors.