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"
}
```

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

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

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

It's a terrible idea because of future maintenance.

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

Use react instead of receive, which never returns.

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

- 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.

Share-nothing, message passing.

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!

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

Send a copy instead.

Shared data and locks.

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

- 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.

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.

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

`scala.actors.Actor.actor factory method`

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

block.

not interrupted.

Use `Actor.self`.

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

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

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