Send and Sync

Not everything obeys inherited mutability, though. Some types allow you to
a location in memory while mutating it. Unless these types use synchroniza
access, they are absolutely not thread-safe. Rust captures this through the

- A type is Send if it is safe to send it to another thread.
- A type is Sync if it is safe to share between threads (`&T` is Send).

Send and Sync are fundamental to Rust's concurrency story. As such, a sub
tooling exists to make them work right. First and foremost, they're unsafe t
they are unsafe to implement, and other unsafe code can assume that they
implemented. Since they're *marker traits* (they have no associated items lik
implemented simply means that they have the intrinsic properties an imple
Incorrectly implementing Send or Sync can cause Undefined Behavior.

Send and Sync are also automatically derived traits. This means that, unlike
type is composed entirely of Send or Sync types, then it is Send or Sync. Alr
Send and Sync, and as a consequence pretty much all types you'll ever inte
Sync.

Major exceptions include:

- raw pointers are neither Send nor Sync (because they have no safety
- `UnsafeCell` isn't Sync (and therefore `Cell` and `RefCell` aren't).
- `Rc` isn't Send or Sync (because the refcount is shared and unsynchro

`Rc` and `UnsafeCell` are very fundamentally not thread-safe: they enable u
mutable state. However raw pointers are, strictly speaking, marked as thre
lint. Doing anything useful with a raw pointer requires dereferencing it, whi
that sense, one could argue that it would be "fine" for them to be marked a

However it's important that they aren't thread-safe to prevent types that co
automatically marked as thread-safe. These types have non-trivial untracke
unlikely that their author was necessarily thinking hard about thread safety
have a nice example of a type that contains a `*mut` that is definitely not th

Types that aren't automatically derived can simply implement them if desir

```
struct MyBox(*mut u8);

unsafe impl Send for MyBox {}
unsafe impl Sync for MyBox {}
```

In the *incredibly rare* case that a type is inappropriately automatically derive
then one can also unimplement Send and Sync:

```
#![feature(optin_builtin_traits)]

// I have some magic semantics for some synchronization primitive
struct SpecialThreadToken(u8);

impl !Send for SpecialThreadToken {}
impl !Sync for SpecialThreadToken {}
```

Note that *in and of itself* it is impossible to incorrectly derive Send and Sync
ascribed special meaning by other unsafe code can possible cause trouble
or Sync.

Most uses of raw pointers should be encapsulated behind a sufficient abst
Sync can be derived. For instance all of Rust's standard collections are Send
contain Send and Sync types) in spite of their pervasive use of raw pointers

and complex ownership. Similarly, most iterators into these collections are
they largely behave like an `&` or `&mut` into the collection.

TODO: better explain what can or can't be Send or Sync. Sufficient to appe