Per RFC 401, if you have a function declaration `foo`:

```
struct S;

// For the purposes of this explanation, all of these
// different kinds of `fn` declarations are equivalent:

fn foo(x: S) { /* ... */ }
# #[cfg(for_demonstration_only)]
extern "C" {
    fn foo(x: S);
}
# #[cfg(for_demonstration_only)]
impl S {
    fn foo(self) { /* ... */ }
}
```

the type of `foo` is **not** `fn(S)`, as one might expect. Rather, it is a unique, zero-sized marker type written here as `typeof(foo)`. However, `typeof(foo)` can be *coerced* to a function pointer `fn(S)`, so you rarely notice this:

```
# struct S;
# fn foo(_: S) {}
let x: fn(S) = foo; // OK, coerces
```

The reason that this matter is that the type `fn(S)` is not specific to any particular function: it's a function *pointer*. So calling `x()` results in a virtual call, whereas `foo()` is statically dispatched, because the type of `foo` tells us precisely what function is being called.

As noted above, coercions mean that most code doesn't have to be concerned with this distinction. However, you can tell the difference when using **transmute** to convert a fn item into a fn pointer.

This is sometimes done as part of an FFI:

```
extern "C" fn foo(userdata: Box<i32>) {
    /* ... */
}

# fn callback(_: extern "C" fn(*mut i32)) {}
# use std::mem::transmute;
unsafe {
    let f: extern "C" fn(*mut i32) = transmute(foo);
    callback(f);
}
```

Here, transmute is being used to convert the types of the fn arguments. This pattern is incorrect because, because the type of `foo` is a function **item** (`typeof(foo)`), which is zero-sized, and the target type (`fn()`) is a function

pointer, which is not zero-sized. This pattern should be rewritten. There are a few possible ways to do this:

- change the original fn declaration to match the expected signature, and do the cast in the fn body (the preferred option)
- cast the fn item of a fn pointer before calling transmute, as shown here:

```
# extern "C" fn foo(_: Box<i32>) {}
# use std::mem::transmute;
# unsafe {
let f: extern "C" fn(*mut i32) = transmute(foo as extern "C" fn(_));
let f: extern "C" fn(*mut i32) = transmute(foo as usize); // works too
# }
```

The same applies to transmutes to `*mut fn()`, which were observed in practice. Note though that use of this type is generally incorrect. The intention is typically to describe a function pointer, but just `fn()` alone suffices for that. `*mut fn()` is a pointer to a fn pointer. (Since these values are typically just passed to C code, however, this rarely makes a difference in practice.)