Sometimes a struct field, function, type, or even a whole package becomes redundant or unnecessary, but must be kept for compatibility with existing programs.

To signal that an identifier should not be used, add a paragraph to its doc comment that begins with `Deprecated:` followed by some information about the deprecation, and a recommendation on what to use instead, if applicable.

The paragraph does not have to be the last paragraph in the doc comment.

Some tools will warn on use of deprecated identifiers and their docs are hidden on pkg.go.dev now that #40850 is implemented.

The original issue to document the "Deprecated" convention was issue #10909.

Note that, if function `F1` is being replaced by function `F2`, and the first release in which `F2` is available is Go 1.N, then we can add an official deprecation notice in release Go 1.N+2. This ensures that Go developers only see `F1` as deprecated when all supported Go versions include `F2` and they can easily switch.

Examples

```
type ResponseRecorder struct {
    // HeaderMap contains the headers explicitly set by the Handler.
    // It is an internal detail.
    //
    // Deprecated: HeaderMap exists for historical compatibility
    // and should not be used. To access the headers returned by a handler,
    // use the Response.Header map as returned by the Result method.
    HeaderMap http.Header
```

```
// Package rc4 implements the RC4 stream cipher.
//
// Deprecated: RC4 is cryptographically broken and should not be used
// except for compatibility with legacy systems.
//
// This package is frozen and no new functionality will be added.
package rc4
```

There are a few other examples in the standard library.