Errors

Errors are indicated by returning an `error` as an additional return value from a function. A `nil` value means that there was no error.

`error`s can be turned into strings by calling `Error`, their only method. You can create an error from a string by calling `errors.New`:

```
if failure {
    return errors.New("inverse tachyon pulse failed")
}
```

or by using `fmt.Errorf`:

```
if failure {
    return fmt.Errorf("inverse tachyon pulse failed")
}
```

Error strings should not start with a capital letter because they'll often be prefixed before printing:

```
err := TryInverseTachyonPulse()
if err != nil {
    fmt.Printf("failed to solve problem: %s\n", err)
}
```

If you expect calling code to be able to handle an error, you can distinguish classes of errors either by returning special values, or new types. You only need to distinguish differences that the calling code could be expected to handle in this way as the string allows one to communicate the details of the error.

`io.EOF` is a special value that signals the end of a stream. You can compare error values directly against io.EOF.

If you want to carry extra data with the error, you can use a new type:

```
type ParseError struct {
    Line, Col int
}

func (p ParseError) Error() string {
    return fmt.Sprintf("parse error on line %d, column %d", p.Line, p.Col)
}
```

If you want to create a constant string error, you can use a named type string:

```
type errorConst string

const ErrTooManyErrors errorConst = "too many errors found."

func (e errorConst) Error() string {
```

```
    return string(e)
}
```

Calling code would test for a special type of `error` by using a type switch:

```
switch err := err.(type) {
case ParseError:
    PrintParseError(err)
}
```

Naming

Error types end in `"Error"` and error variables start with `"Err"` or `"err"`:

```
package somepkg

// ParseError is type of error returned when there's a parsing problem.
type ParseError struct {
  Line, Col int
}

var ErrBadAction = errors.New("somepkg: a bad action was performed")

// -----

package foo

func foo() {
    res, err := somepkgAction()
    if err != nil {
        if err == somepkg.ErrBadAction {
        }
        if pe, ok := err.(*somepkg.ParseError); ok {
             line, col := pe.Line, pe.Col
             // ....
        }
    }
}
```

References

- Errors (specification): https://go.dev/ref/spec#Errors
- Package `errors`: https://pkg.go.dev/errors/
- Type switches: https://go.dev/ref/spec#TypeSwitchStmt