Error Handling

01

Error Interface

02

Make Your Own Errors

03

Error Checking

Error Handling

- Go has no exceptions
- Errors are returned as the **last** return value from a function
  - Encodes failure as part of the function signature
    - Simple to determine if a function can fail
  - Return **nil** if no error occurred
- Errors implement the **error** interface from **std**
  - One function to implement: **Error() string**

Basics

- The **errors** stdlib package can generate simple errors with the **New()** function

```
import "errors"

func divide(lhs, rhs int) (int, error) {
    if rhs == 0 {
        return 0, errors.New("cannot divide by zero")
    } else {
        return rhs / lhs, nil
    }
}
```

Error Interface

```go
type error interface {
    Error() string
}
```

Implementation

```go
type DivError struct {
    a, b int
}

func (d *DivError) Error() string {
    return fmt.Sprintf("Cannot divide by zero: %d / %d", d.a, d.b)
}
```

- Always implement **error** as a **receiver function**
  - Prevents comparison problems if error is inspected

Usage

```go
type DivError struct {
    a, b int
}

func div(a, b int) (int, error) {
    if b == 0 {
        return 0, &DivError{a, b}
    } else {
        return a / b, nil
    }
}

answer1, err := div(9, 0)
if err != nil {
    // "Cannot divide by zero: 9 / 0"
    fmt.Println(err)
    return
}
fmt.Println("The answer is:", answer1)
```

Working With Errors

- Use **errors.Is()** to determine if an error contains a specific type

```go
type UserError struct {
    Msg string
}

func (u *UserError) Error() string {
    return fmt.Sprintf("User error: %v", string(u.Msg))
}

_, err := someFunc("sample")
if err != nil {
    var InputError = UserError{"Input Error"}
    if errors.Is(err, &InputError) {
        fmt.Println("Input error:", err)
    } else {
        fmt.Println("Other error:", err)
    }
}
```

Working With Errors

- Use **errors.As()** to retrieve a specific error

```go
type UserError struct {
    Msg string
}

func (u *UserError) Error() string {
    return fmt.Sprintf("User error: %v", string(u.Msg))
}

_, err := someFunc("sample")
if err != nil {
    var thisError *UserError
    if errors.As(err, &thisError) {
        fmt.Println("User error:", thisError)
    } else {
        fmt.Println("Other error:", err)
    }
}
```

Recap

- Errors are returned as the **last** return value from a function
- Use **errors.New()** to generate simple errors
  - Use **errors.As()** to retrieve an error, or **errors.Is()** to check the error type
- Implement the **error** interface for custom errors
  - Always implement the interface as a **receiver function**

```
type error interface {
    Error() string
}
```

- Always check if **err != nil** for functions that return an error type