DWARVES
FOUNDATION

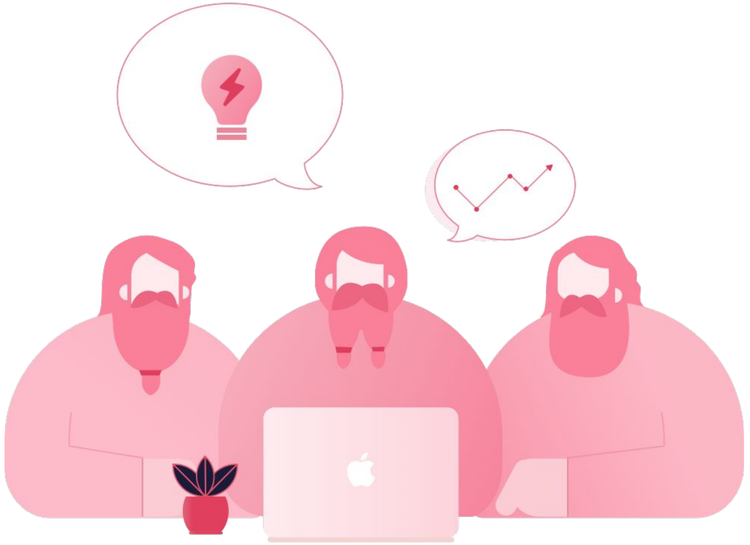
Error Handling and Testing

Author Name

Hieu Phan

@hieupq

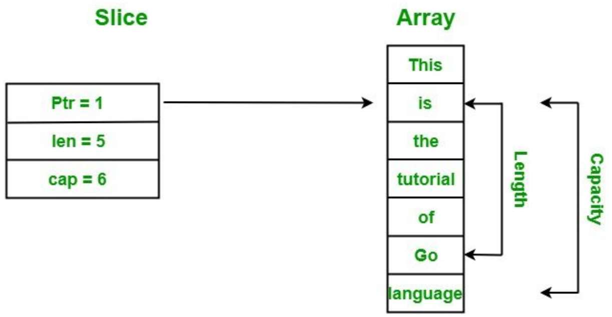
andy@d.foundation



Agenda

1. Introduction
2. Error Handling
3. Unit Testing
4. Demo

Day 3



- Interface & structs
- Pointers store the memory address of a variable's value
- Arrays are fixed-size collections of elements with a specific data type.
- Slices are dynamic, resizable views into arrays
- Maps are key-value pairs that provide efficient data retrieval and storage.

Error

Errors are values.

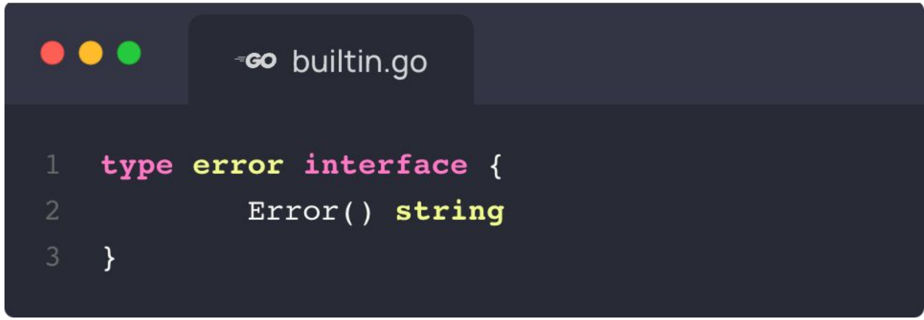
Error

- errors are a first-class citizen, treated as values rather than exceptions.
- Error handling is essential for creating robust and reliable applications.

DWARVES FOUNDATION

Error Interface

Any type that implements this method is considered an error in Go.

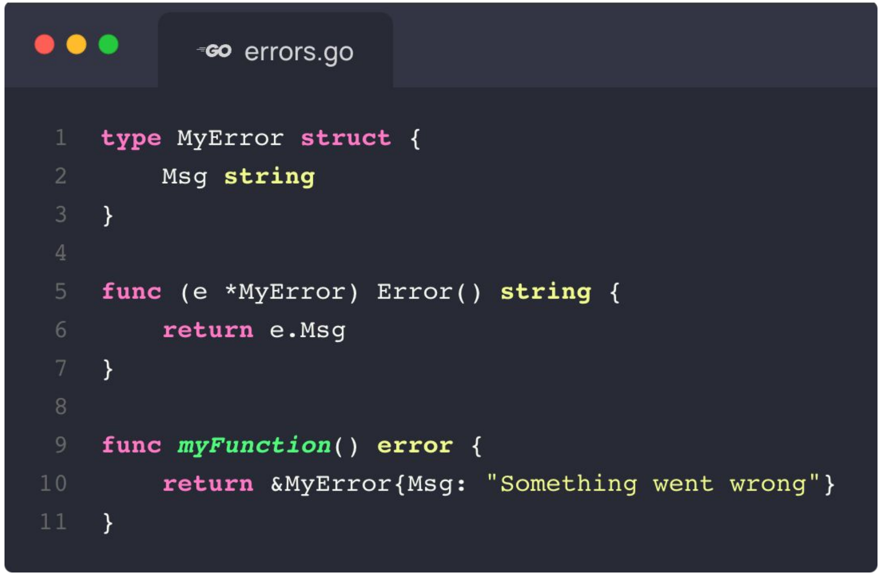


```go
type error interface {
        Error() string
}
```

DWARVES FOUNDATION

Custom Error type

Custom error types can help differentiate different types of errors and make error handling more expressive.

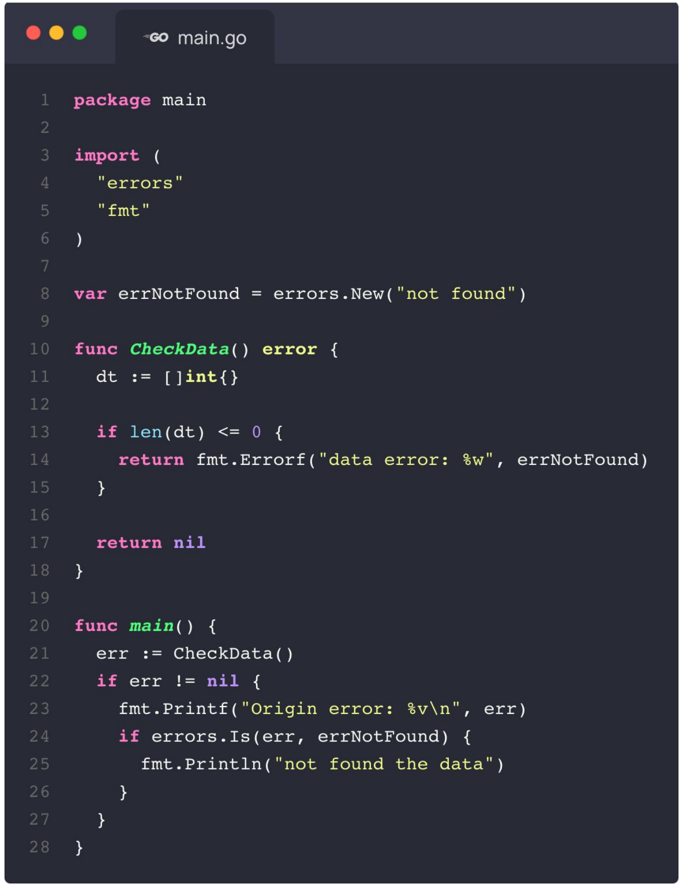


```go
type MyError struct {
    Msg string
}

func (e *MyError) Error() string {
    return e.Msg
}

func myFunction() error {
    return &MyError{Msg: "Something went wrong"}
}
```

DWARVES FOUNDATION

New

- **New** function returns a new error that formats as the given text
- Go 1.13, the **fmt.Errorf** function supports a new %w verb.

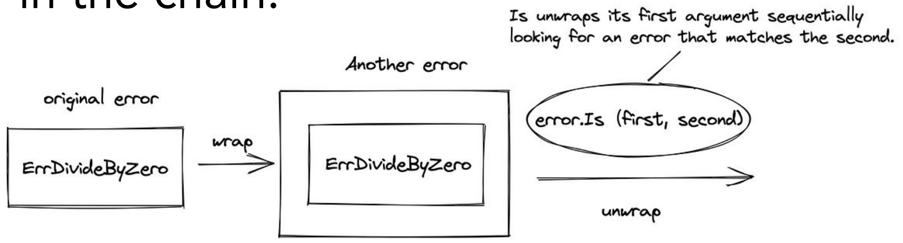

main.go

```go
package main

import (
  "errors"
  "fmt"
)

var errNotFound = errors.New("not found")

func CheckData() error {
  dt := []int{}

  if len(dt) <= 0 {
    return fmt.Errorf("data error: %w", errNotFound)
  }

  return nil
}

func main() {
  err := CheckData()
  if err != nil {
    fmt.Printf("Origin error: %v\n", err)
    if errors.Is(err, errNotFound) {
      fmt.Println("not found the data")
    }
  }
}
```

DWARVES FOUNDATION

Unwrap error

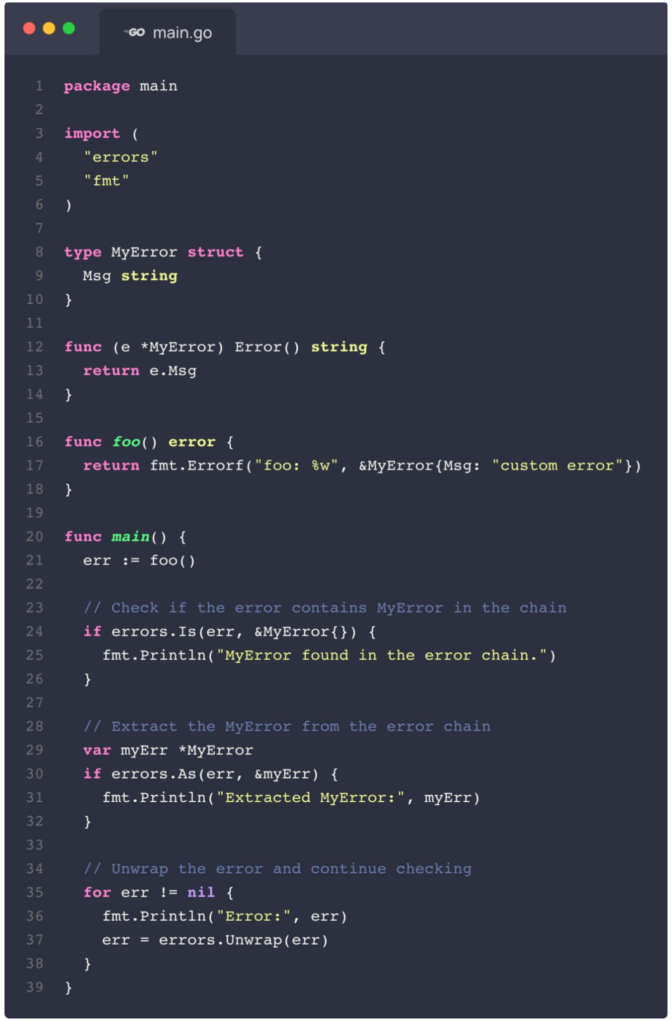
- The **Is()** function reports whether any error in the chain of err matches the target.
- The **As()** function attempts to find the first error in the error chain that can be assigned to the type of target.
- The **Unwrap()** function returns the next error in the error chain, or nil if there is no more error in the chain.



DWARVES FOUNDATION

Error package Practices

- Prefer Specific Error Types
- Use fmt.Errorf for Wrapping Errors
- Utilize errors.Is() and errors.As()
- Avoid Silent Error Handling


main.go

```go
package main

import (
  "errors"
  "fmt"
)

type MyError struct {
  Msg string
}

func (e *MyError) Error() string {
  return e.Msg
}

func foo() error {
  return fmt.Errorf("foo: %w", &MyError{Msg: "custom error"})
}

func main() {
  err := foo()

  // Check if the error contains MyError in the chain
  if errors.Is(err, &MyError{}) {
    fmt.Println("MyError found in the error chain.")
  }

  // Extract the MyError from the error chain
  var myErr *MyError
  if errors.As(err, &myErr) {
    fmt.Println("Extracted MyError:", myErr)
  }

  // Unwrap the error and continue checking
  for err != nil {
    fmt.Println("Error:", err)
    err = errors.Unwrap(err)
  }
}
```

DWARVES FOUNDATION

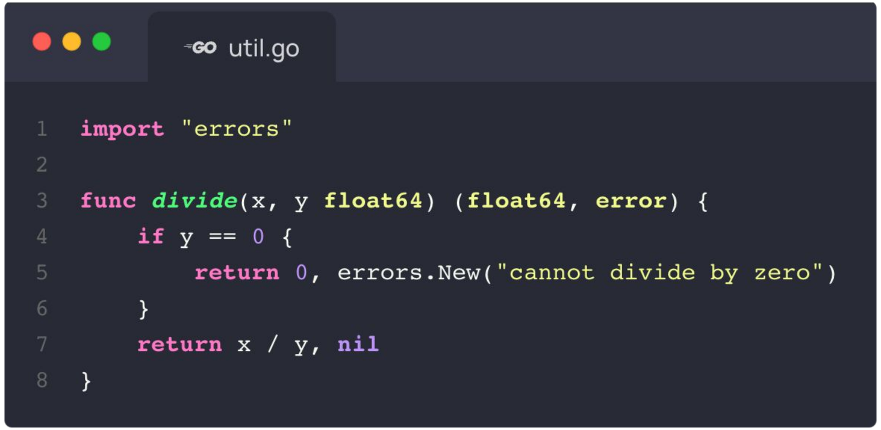
Error Handling

Error in function

DWARVES FOUNDATION

Error in function

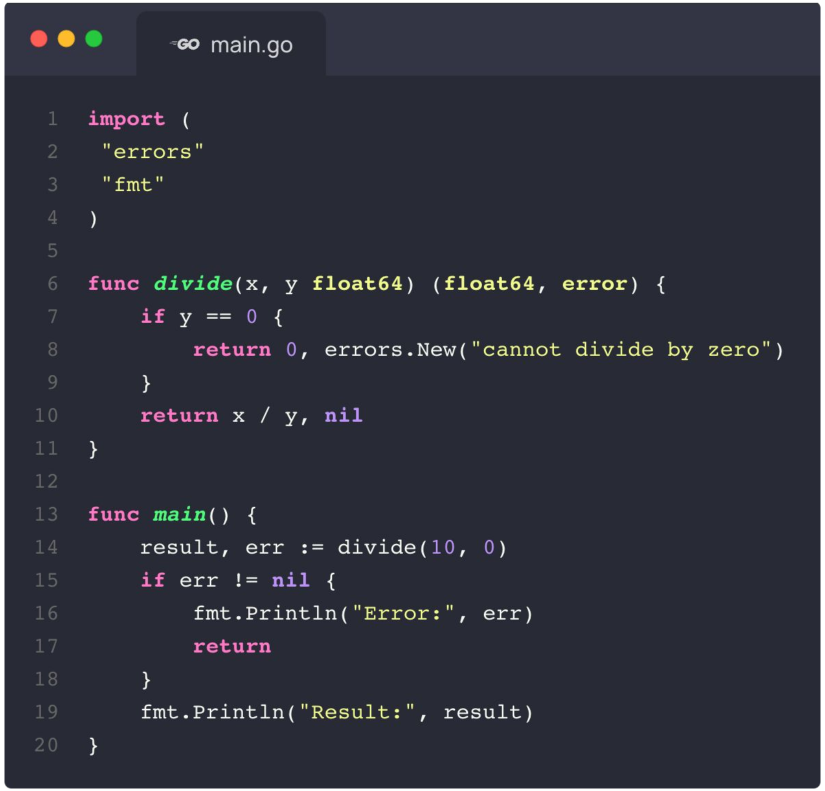
- Functions in Go that can encounter errors typically return two values, where the second value is the error.
- Conventionally, the last return value is the error, and it is set to **nil** if the function executes successfully without any errors.



```go
import "errors"

func divide(x, y float64) (float64, error) {
    if y == 0 {
        return 0, errors.New("cannot divide by zero")
    }
    return x / y, nil
}
```

Handling the result

- After calling a function that returns an error, it is essential to check the error value.
- If the error is non-nil, it means an error occurred during the function's execution, and you should handle it appropriately.



```go
import (
 "errors"
 "fmt"
)

func divide(x, y float64) (float64, error) {
    if y == 0 {
        return 0, errors.New("cannot divide by zero")
    }
    return x / y, nil
}

func main() {
    result, err := divide(10, 0)
    if err != nil {
        fmt.Println("Error:", err)
        return
    }
    fmt.Println("Result:", result)
}
```

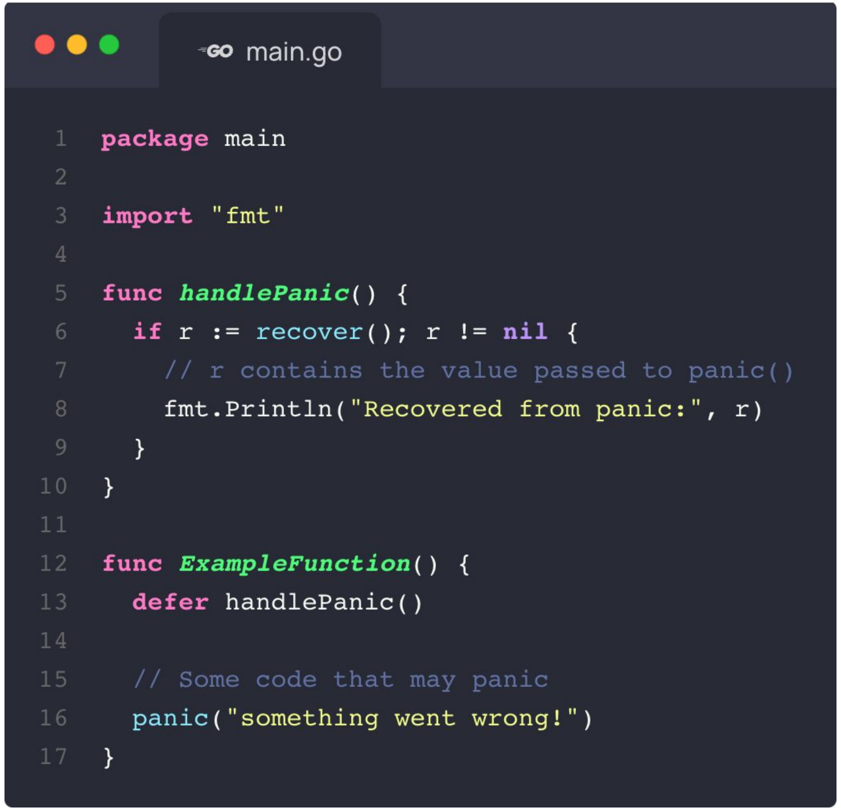
DWARVES FOUNDATION

Panic & Recover

Abruptly terminate the program using the panic keyword

Panic

- When the **panic** keyword is encountered, it immediately stops the normal flow of execution and starts the panic process.

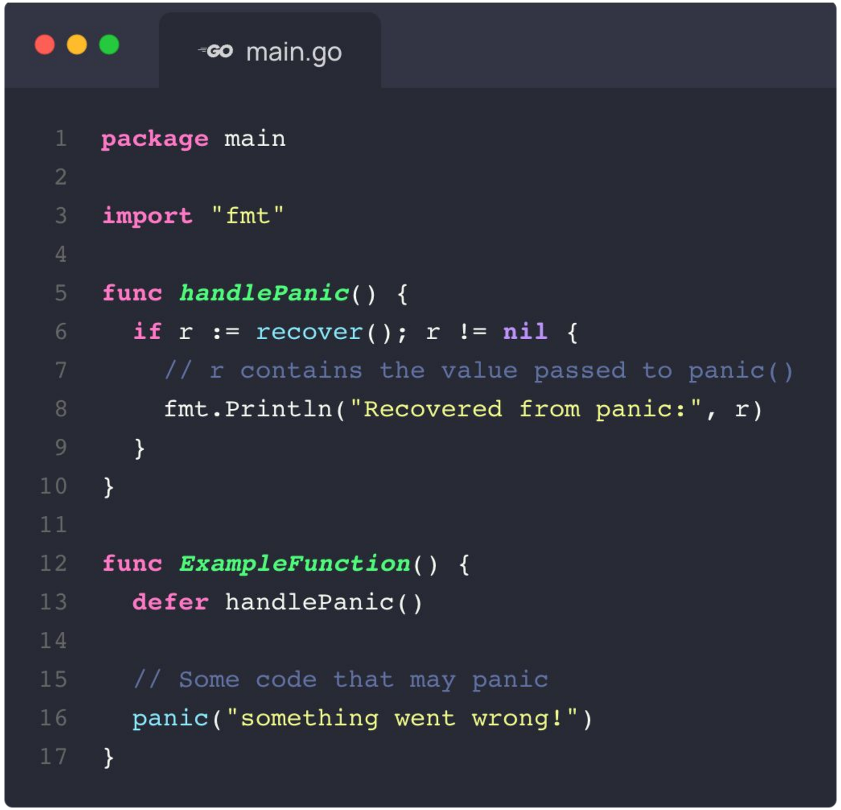


```go
package main

import "fmt"

func handlePanic() {
  if r := recover(); r != nil {
    // r contains the value passed to panic()
    fmt.Println("Recovered from panic:", r)
  }
}

func ExampleFunction() {
  defer handlePanic()

  // Some code that may panic
  panic("something went wrong!")
}
```

DWARVES FOUNDATION

Recover

- The **recover** function is a built-in function used to capture and handle panics.
- It is typically used in deferred functions to intercept and gracefully recover from a panic.


main.go

```go
package main

import "fmt"

func handlePanic() {
  if r := recover(); r != nil {
    // r contains the value passed to panic()
    fmt.Println("Recovered from panic:", r)
  }
}

func ExampleFunction() {
  defer handlePanic()

  // Some code that may panic
  panic("something went wrong!")
}
```

DWARVES FOUNDATION

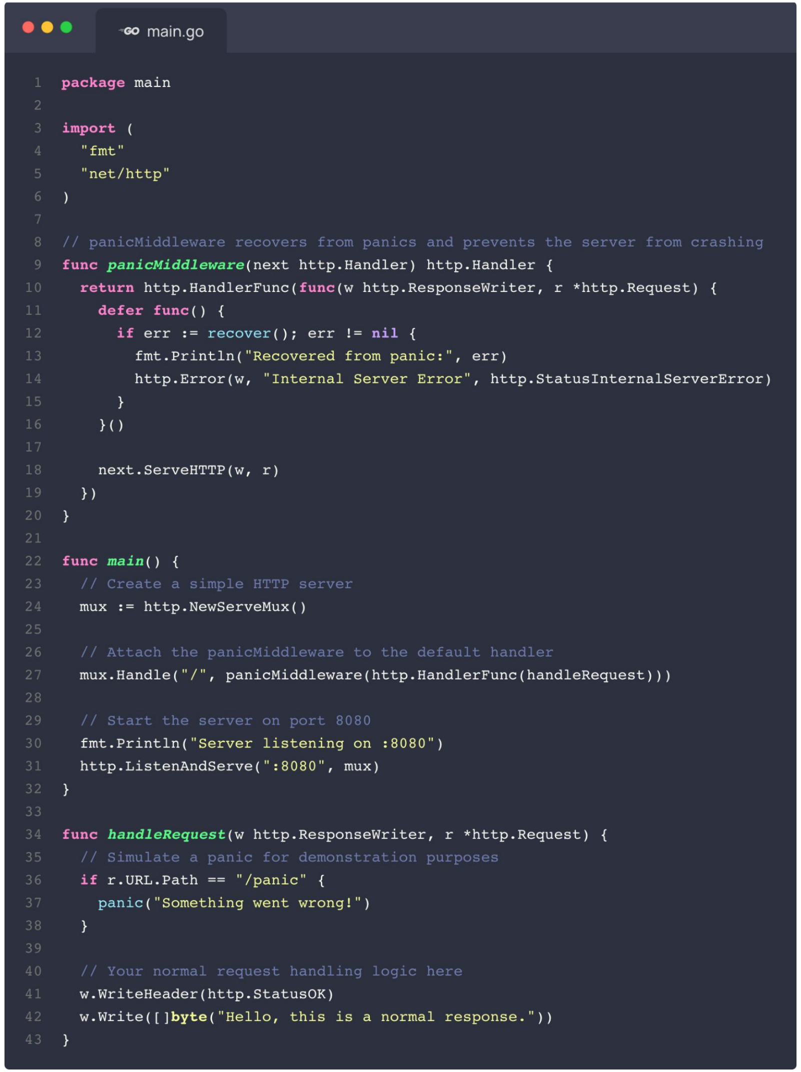
Caution

- panic and recover are powerful tools but …
- They are not meant for routine error handling or normal program flow control.
- Can lead to code that is difficult to maintain and understand.

Practices

- Only use panic in exceptional, unrecoverable situations.
- Avoid using panic as a flow control mechanism or to handle expected errors.
- Always use recover in deferred functions to capture and handle panics when needed.
- Provide clear and informative panic messages to aid in debugging and error analysis.

Practices


main.go

```go
package main

import (
  "fmt"
  "net/http"
)

// panicMiddleware recovers from panics and prevents the server from crashing
func panicMiddleware(next http.Handler) http.Handler {
  return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
    defer func() {
      if err := recover(); err != nil {
        fmt.Println("Recovered from panic:", err)
        http.Error(w, "Internal Server Error", http.StatusInternalServerError)
      }
    }()

    next.ServeHTTP(w, r)
  })
}

func main() {
  // Create a simple HTTP server
  mux := http.NewServeMux()

  // Attach the panicMiddleware to the default handler
  mux.Handle("/", panicMiddleware(http.HandlerFunc(handleRequest)))

  // Start the server on port 8080
  fmt.Println("Server listening on :8080")
  http.ListenAndServe(":8080", mux)
}

func handleRequest(w http.ResponseWriter, r *http.Request) {
  // Simulate a panic for demonstration purposes
  if r.URL.Path == "/panic" {
    panic("Something went wrong!")
  }

  // Your normal request handling logic here
  w.WriteHeader(http.StatusOK)
  w.Write([]byte("Hello, this is a normal response."))
}
```

DWARVES FOUNDATION

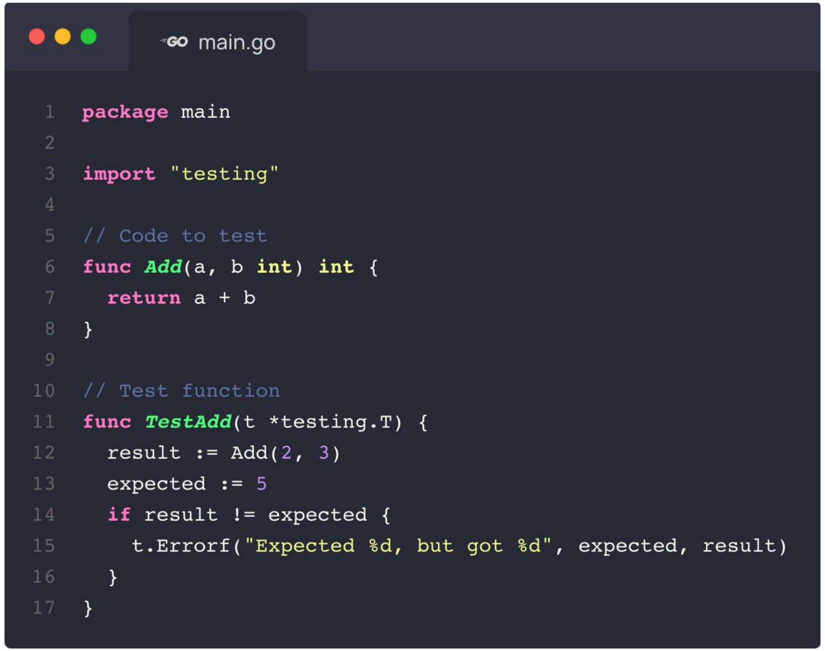
Unit testing

Ensure the correctness and reliability of code

DWARVES
FOUNDATION

Unit test

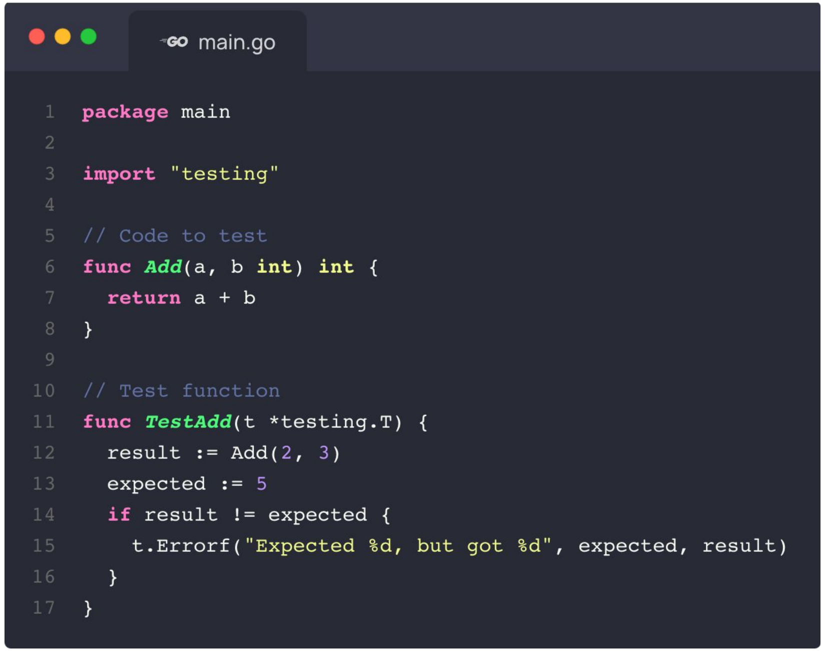
- Test functions in Go start with the word **Test** and accept a single parameter of type ***testing.T**.
- The ***testing.T** parameter provides methods to report test failures and log messages during test execution.



```go
package main

import "testing"

// Code to test
func Add(a, b int) int {
  return a + b
}

// Test function
func TestAdd(t *testing.T) {
  result := Add(2, 3)
  expected := 5
  if result != expected {
    t.Errorf("Expected %d, but got %d", expected, result)
  }
}
```

Run test

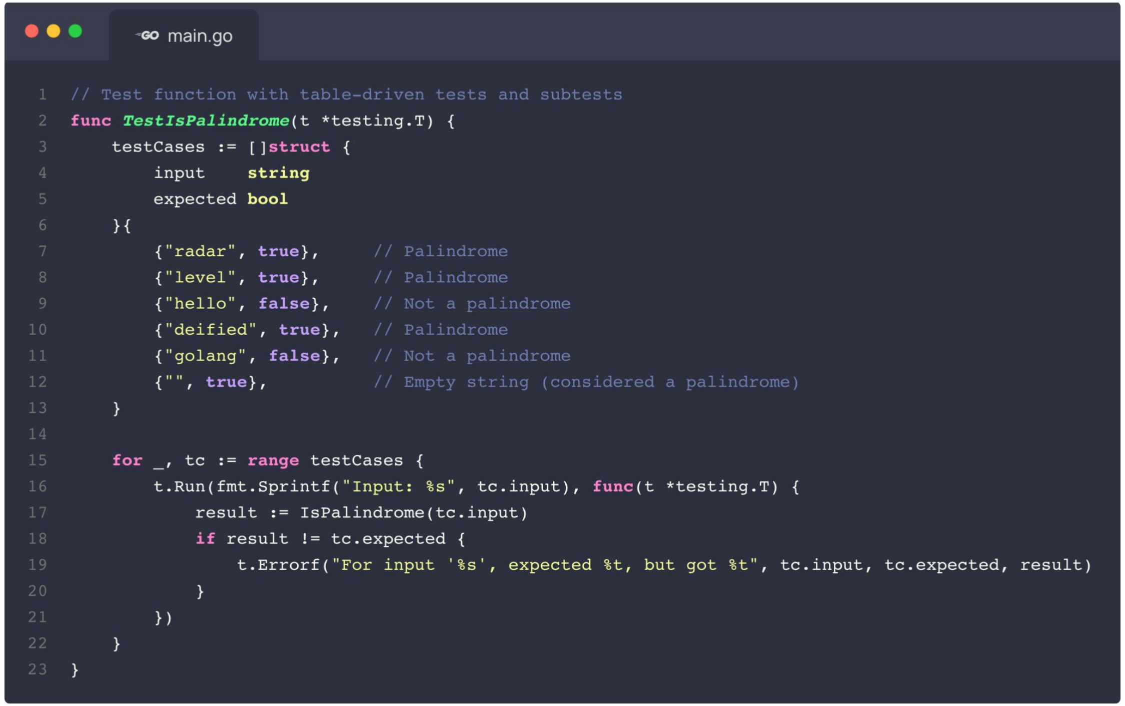
- go test
- go test -cover



```go
package main

import "testing"

// Code to test
func Add(a, b int) int {
  return a + b
}

// Test function
func TestAdd(t *testing.T) {
  result := Add(2, 3)
  expected := 5
  if result != expected {
    t.Errorf("Expected %d, but got %d", expected, result)
  }
}
```

Write unit test

- Table-Driven Tests
- Subtests

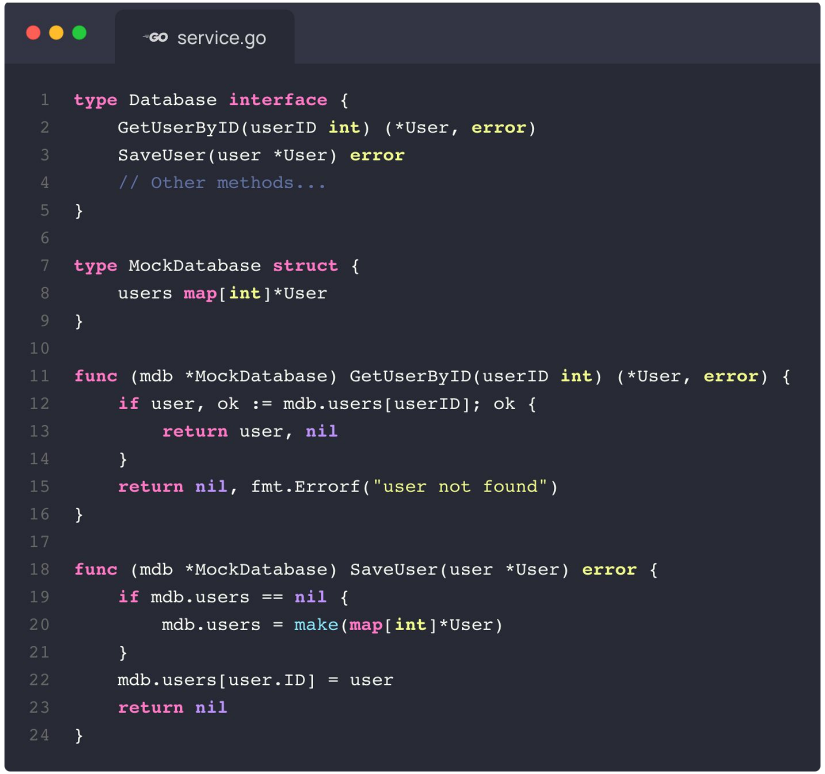


main.go

```go
// Test function with table-driven tests and subtests
func TestIsPalindrome(t *testing.T) {
    testCases := []struct {
        input    string
        expected bool
    }{
        {"radar", true},      // Palindrome
        {"level", true},      // Palindrome
        {"hello", false},     // Not a palindrome
        {"deified", true},    // Palindrome
        {"golang", false},    // Not a palindrome
        {"", true},           // Empty string (considered a palindrome)
    }

    for _, tc := range testCases {
        t.Run(fmt.Sprintf("Input: %s", tc.input), func(t *testing.T) {
            result := IsPalindrome(tc.input)
            if result != tc.expected {
                t.Errorf("For input '%s', expected %t, but got %t", tc.input, tc.expected, result)
            }
        })
    }
}
```

Mocking in unit test

- libraries: gomock, mockery
- httptest for HTTP Testing



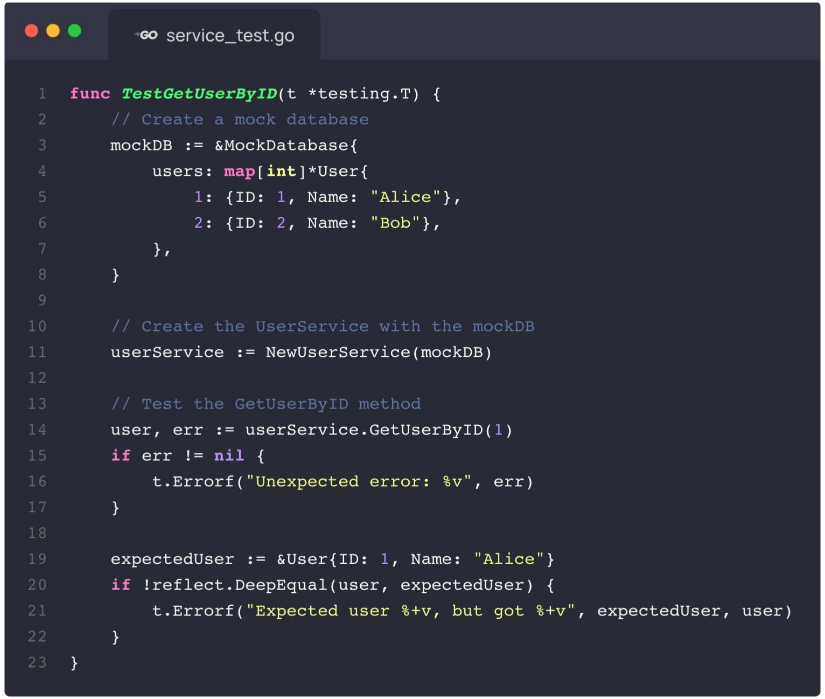
service.go

```go
type Database interface {
    GetUserByID(userID int) (*User, error)
    SaveUser(user *User) error
    // Other methods...
}

type MockDatabase struct {
    users map[int]*User
}

func (mdb *MockDatabase) GetUserByID(userID int) (*User, error) {
    if user, ok := mdb.users[userID]; ok {
        return user, nil
    }
    return nil, fmt.Errorf("user not found")
}

func (mdb *MockDatabase) SaveUser(user *User) error {
    if mdb.users == nil {
        mdb.users = make(map[int]*User)
    }
    mdb.users[user.ID] = user
    return nil
}
```

DWARVES FOUNDATION

Mocking in unit test

https://dwarvesf.hashnode.dev/understanding-test-doubles-an-in-depth-look


service_test.go

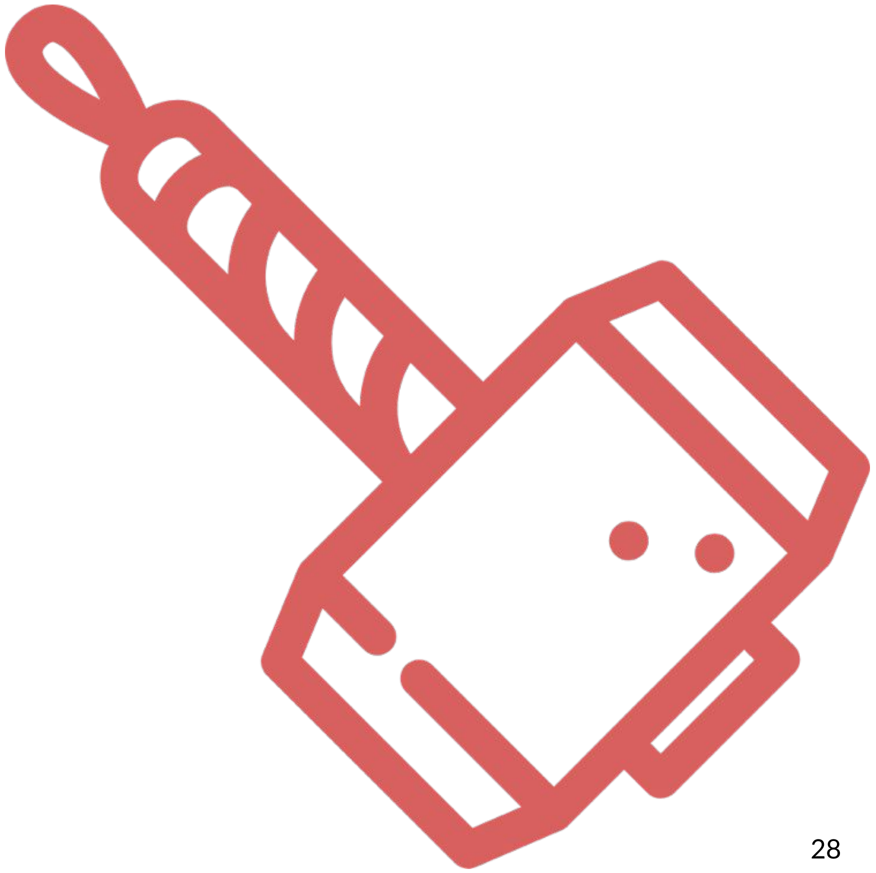
```go
func TestGetUserByID(t *testing.T) {
    // Create a mock database
    mockDB := &MockDatabase{
        users: map[int]*User{
            1: {ID: 1, Name: "Alice"},
            2: {ID: 2, Name: "Bob"},
        },
    }

    // Create the UserService with the mockDB
    userService := NewUserService(mockDB)

    // Test the GetUserByID method
    user, err := userService.GetUserByID(1)
    if err != nil {
        t.Errorf("Unexpected error: %v", err)
    }

    expectedUser := &User{ID: 1, Name: "Alice"}
    if !reflect.DeepEqual(user, expectedUser) {
        t.Errorf("Expected user %+v, but got %+v", expectedUser, user)
    }
}
```

DWARVES FOUNDATION

Demo

Demo - Zer0 to Hero

- Make unit tests with vscode



DWARVES FOUNDATION

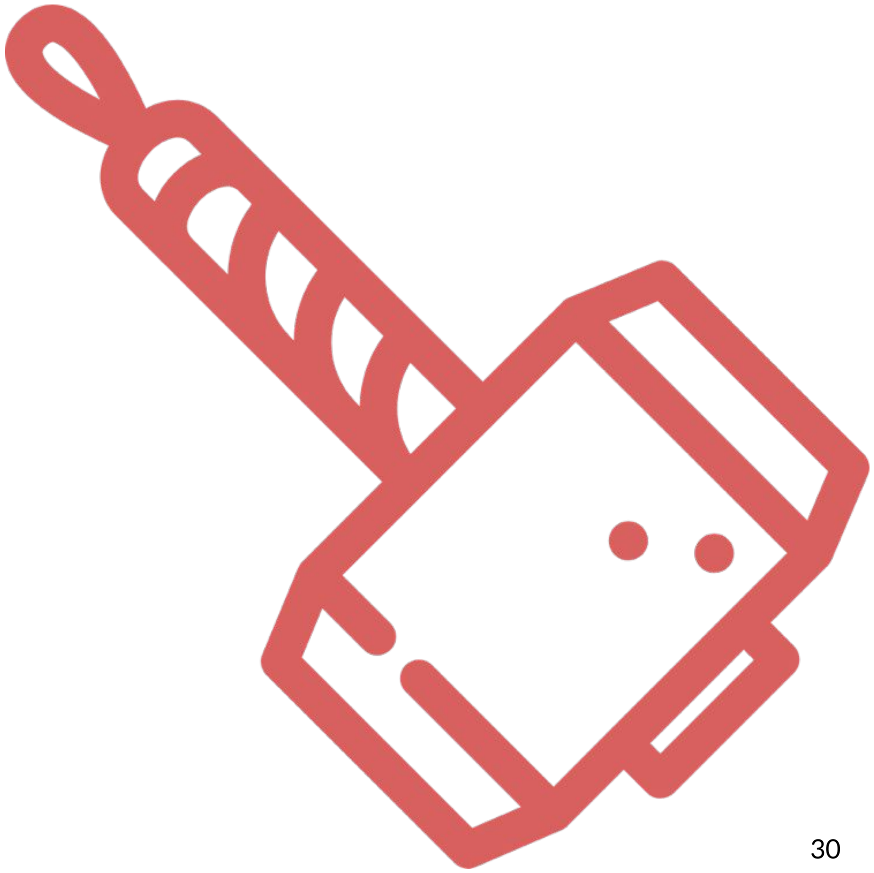
Reference

Resources & Reference links

- https://github.com/golang/go/blob/0e08b02ac54c9232759704812f41a5836f920cff/src/builtin/builtin.go#L280-L282
- https://go.dev/blog/go1.13-errors
- https://dwarvesf.hashnode.dev/error-handling-and-failure-management-in-a-go-system
- https://github.com/DATA-DOG/go-sqlmock

Assignment 4

- Make unit tests for prev project: assignment 3a, 3b



DWARVES FOUNDATION

DWARVES FOUNDATION

Thank You

DWARVES FOUNDATION

Q&A