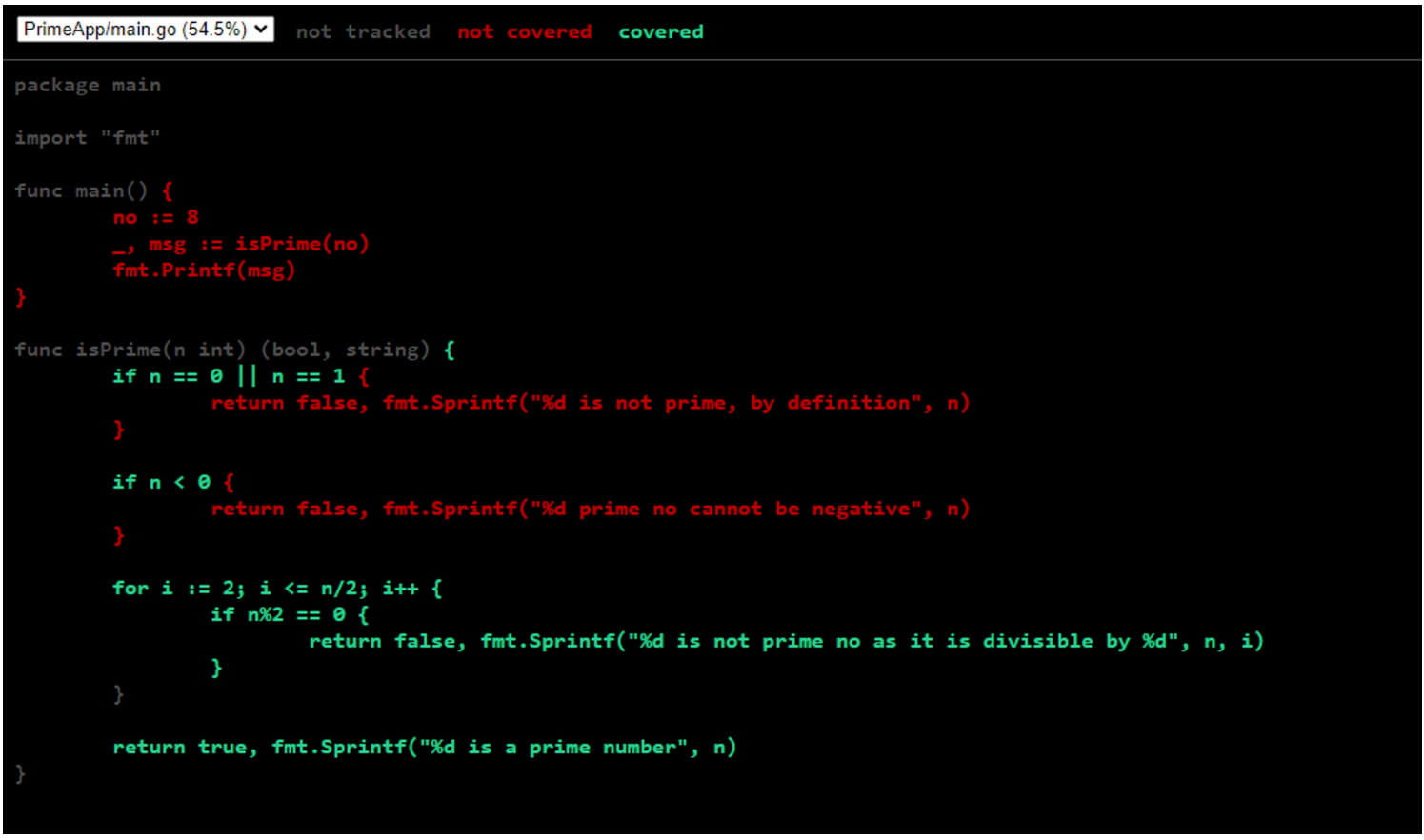
Testing

Tuesday, January 17, 2023 9:33 PM

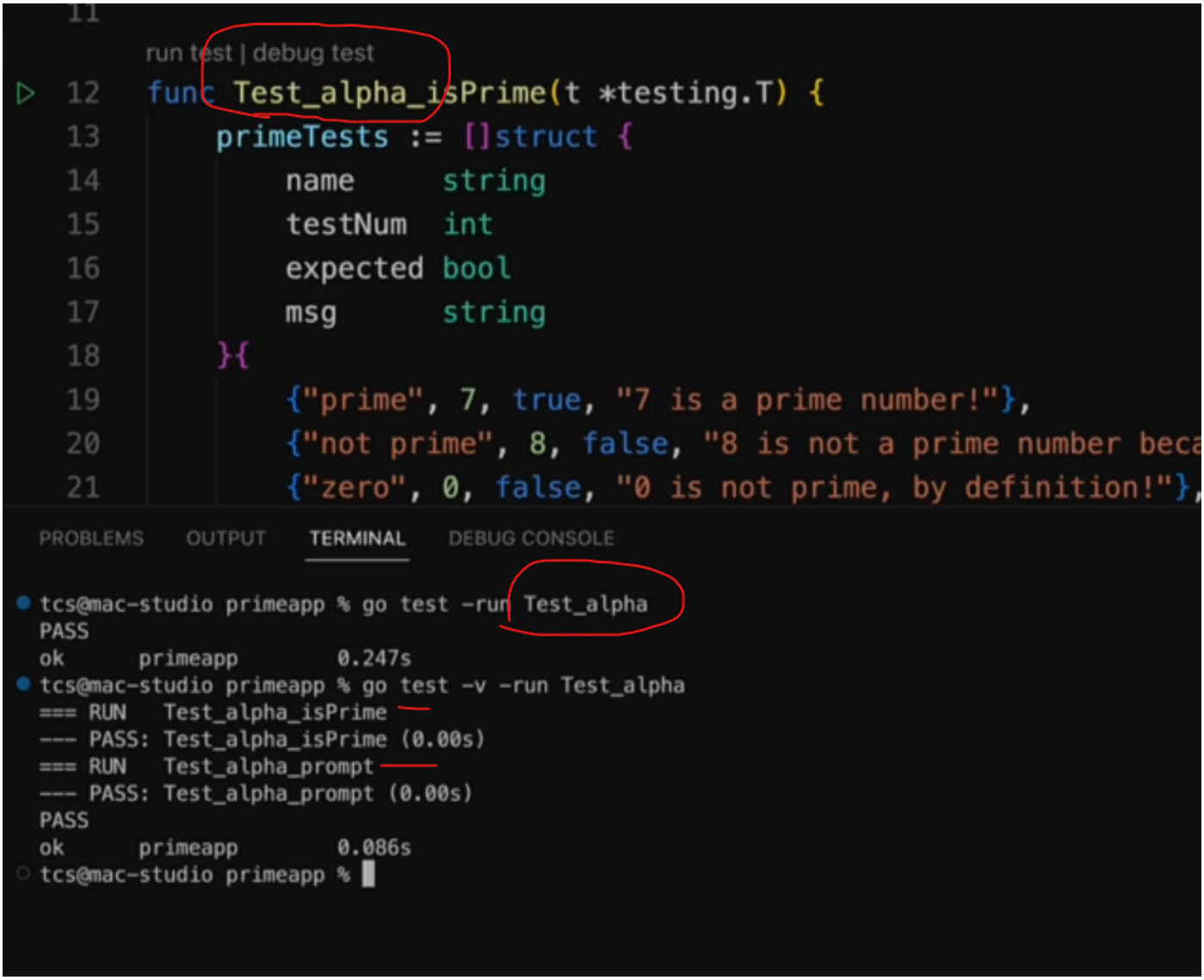
```
Akash@LAPTOP-0S70H5K5 MINGW64 /d/Personal/Programming/Go/Introduction to Testing in Go (Golang)/Testing Practise/PrimeApp
$ go test -cover .
ok      PrimeApp        0.032s  coverage: 54.5% of statements

Akash@LAPTOP-0S70H5K5 MINGW64 /d/Personal/Programming/Go/Introduction to Testing in Go (Golang)/Testing Practise/PrimeApp
$ go test -coverprofile=coverage.out
PASS
coverage: 54.5% of statements
ok      PrimeApp        0.037s

Akash@LAPTOP-0S70H5K5 MINGW64 /d/Personal/Programming/Go/Introduction to Testing in Go (Golang)/Testing Practise/PrimeApp
$ go tool cover -html=coverage.out
```



go test -coverprofile=coverage.out && go tool cover -html=coverage.out

To run only one test

To run group of test (test suites)



All the test cases which has naming convention as Test_alpha will run.

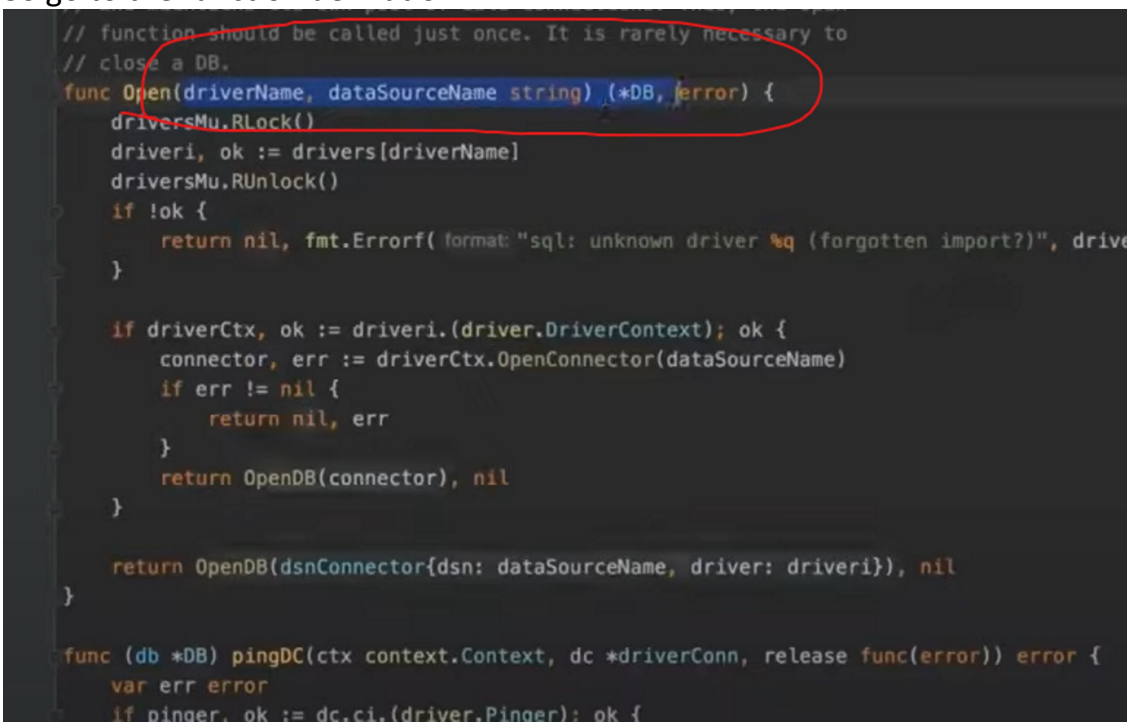
Learn by ~~example~~ situation

I hate long examples to explain simple concepts…

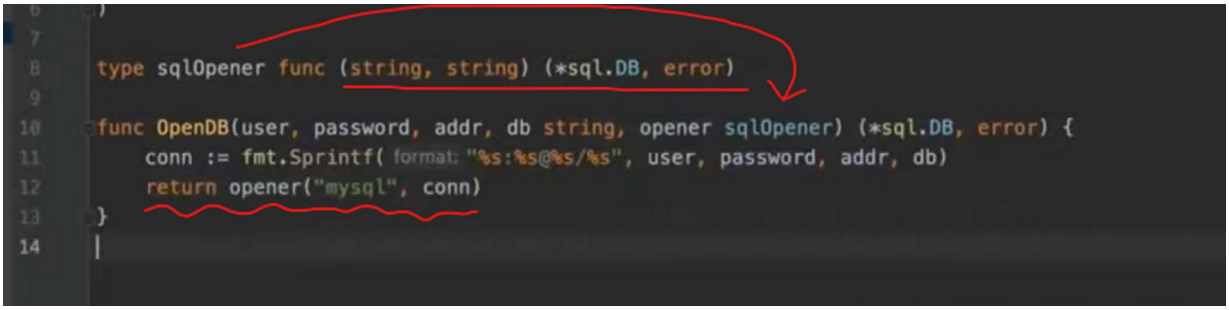
- “I need to mock a function”
- “I need to mock a method on a type”
- “I have a large interface, I need to mock a small set of its methods”
- “I need to mock an HTTP call”

```
)

func OpenDB(user, password, addr, db string) (*sql.DB, error) {
    conn := fmt.Sprintf( format: "%s:%s@%s/%s", user, password, addr, db)
    return sql.Open( driverName: "mysql", conn)
}
```

This is the trouble maker
So go to the function defination



```
// function should be called just once. It is rarely necessary to
// close a DB.
func Open(driverName, dataSourceName string) (*DB, error) {
    driversMu.RLock()
    driveri, ok := drivers[driverName]
    driversMu.RUnlock()
    if !ok {
        return nil, fmt.Errorf( format: "sql: unknown driver %q (forgotten import?)", drive
    }

    if driverCtx, ok := driveri.(driver.DriverContext); ok {
        connector, err := driverCtx.OpenConnector(dataSourceName)
        if err != nil {
            return nil, err
        }
        return OpenDB(connector), nil
    }

    return OpenDB(dsnConnector{dsn: dataSourceName, driver: driveri}), nil
}

func (db *DB) pingDC(ctx context.Context, dc *driverConn, release func(error)) error {
    var err error
    if pinger, ok := dc.ci.(driver.Pinger); ok {
```



```go
type sqlOpener func (string, string) (*sql.DB, error)

func OpenDB(user, password, addr, db string, opener sqlOpener) (*sql.DB, error) {
    conn := fmt.Sprintf( format: "%s:%s@%s/%s", user, password, addr, db)
    return opener("mysql", conn)
}
```

```go
func TestOpenDB(t *testing.T) {
    mockError := errors.New( text: "uh oh")
    subtests := []struct {
        name        string
        u, p, a, db string
        sqlOpener   func(string, string) (*sql.DB, error)
        expectedErr error
    }{
        {
            name: "happy-path",
            u:    "u",
            p:    "p",
            a:    "a",
            db:   "db",
            sqlOpener: func(driver string, source string) (db *sql.DB, err error) {
                if source != "u:p@a/db" {
                    return  db: nil, errors.New( text: "wrong connection string")
                }
                return  db: nil,  err: nil
            },
        },
        {
            name: "error-from-sqlOpener",
            sqlOpener: func(driver string, source string) (db *sql.DB, err error) {
                return  db: nil, mockError
            },
            expectedErr: mockError,
        },
    }
    for _, subtest := range subtests {
        t.Run(subtest.name, func(t *testing.T) {
            _, err := sqlexample.OpenDB(subtest.u, subtest.p, subtest.a, subtest.db, subtest.sqlOpener)
            if !errors.Is(err, subtest.expectedErr) {
                t.Errorf( format: "expected error (%v), got error (%v)", subtest.expectedErr, err)
            }
        })
    }
}
```

Higher Order Functions

Good

- Clear and proximal to function under test
- Stateless

Not so good....

- Parameter list can get ugly (using a *type* for the function can help)
- Think of dependency graph

Monkey Patching

```go
func TestOpenDB(t *testing.T) {
    mockError := errors.New( text: "uh oh")
    subtests := []struct {
        name        string
        u, p, a, db string
        sqlOpener   func(string, string) (*sql.DB, error)
        expectedErr error
    }{
        {
            name: "happy-path",
            u:    "u",
            p:    "p",
            a:    "a",
            db:   "db",
            sqlOpener: func(driver string, source string) (db *sql.DB, err error) {
                if source != "u:p@a/db" {
                    return  db: nil, errors.New( text: "wrong connection string")
                }
                return  db: nil,  err: nil
            },
        },
        {
            name: "error-from-sqlOpener",
            sqlOpener: func(driver string, source string) (db *sql.DB, err error) {
                return  db: nil, mockError
            },
            expectedErr: mockError,
        },
    }
    for _, subtest := range subtests {
        t.Run(subtest.name, func(t *testing.T) {
            sqlexample.SQLOpen = subtest.sqlOpener
            _, err := sqlexample.OpenDB(subtest.u, subtest.p, subtest.a, subtest.db)
            if !errors.Is(err, subtest.expectedErr) {
                t.Errorf( format: "expected error (%v), got error (%v)", subtest.expectedErr, err)
            }
        })
    }
}
```

Monkey Patching

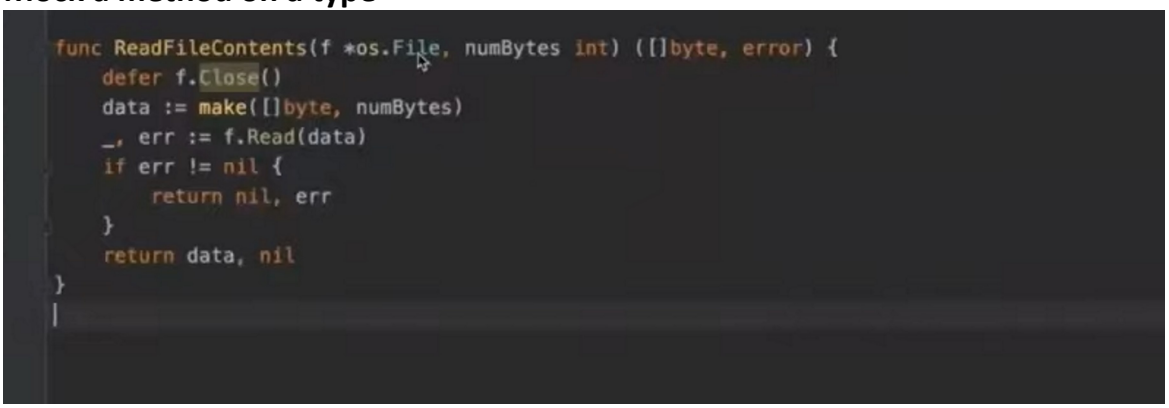
Good

- Don’t need to modify function signature

Not so good...

- Allergic to parallelism (stateful)
- Have to make variable public (if testing from _test package)

Mock a method on a type



```
func ReadFileContents(f *os.File, numBytes int) ([]byte, error) {
    defer f.Close()
    data := make([]byte, numBytes)
    _, err := f.Read(data)
    if err != nil {
        return nil, err
    }
    return data, nil
}
```

So here we need to mock a file
We don’t want to do the operation on the file

1) Checking the file implementation

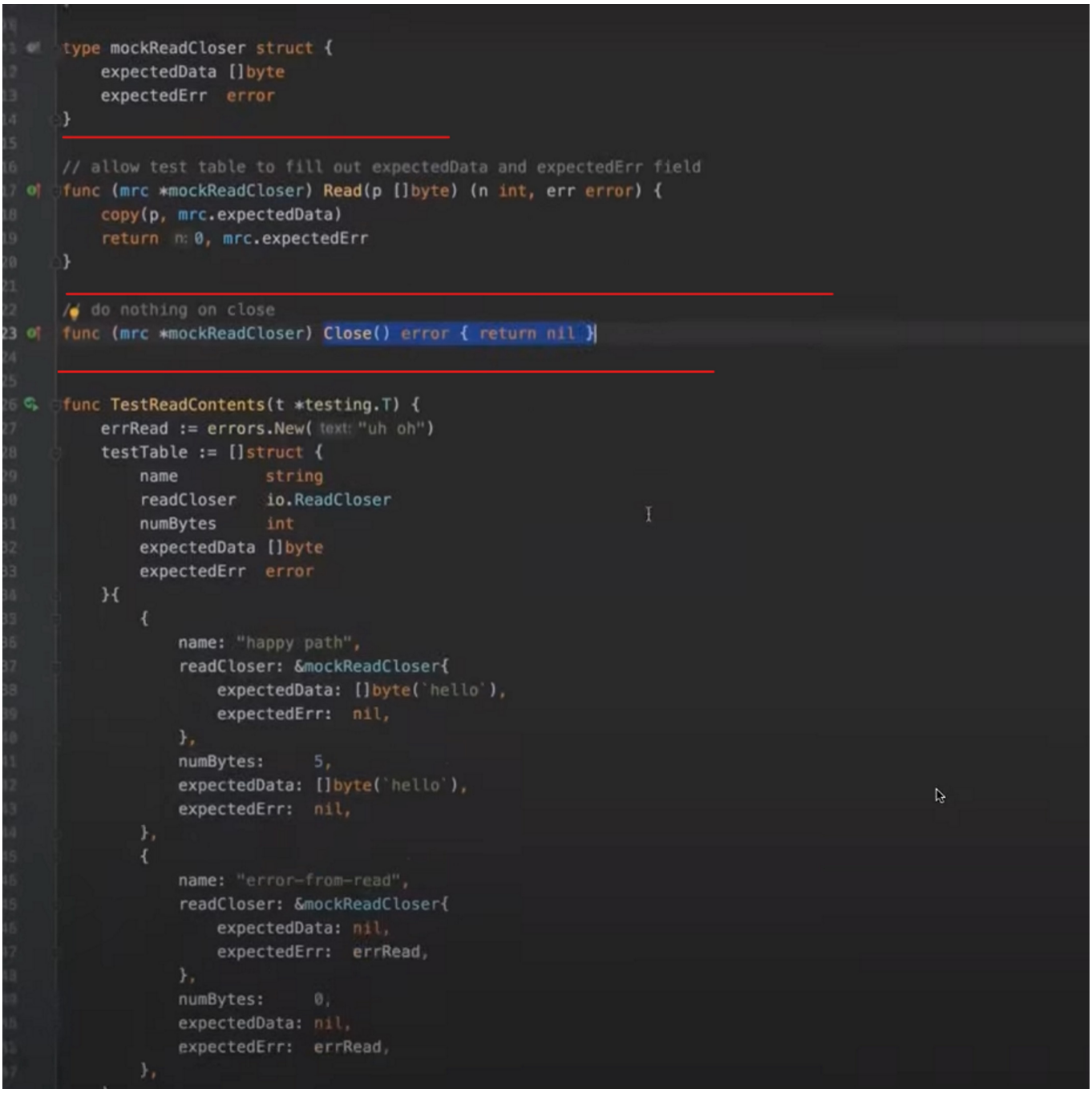
```
func Getpagesize() int { return syscall.Getpagesize() }

// File represents an open file descriptor.
type File struct {
    *file // os specific
}

// A FileInfo describes a file and is returned by Stat and Lstat.
type FileInfo interface {
    Name() string       // base name of the file
    Size() int64        // length in bytes for regular files; system-dependent for other
    Mode() FileMode     // file mode bits
    ModTime() time.Time // modification time
    IsDir() bool        // abbreviation for Mode().IsDir()
    Sys() interface{}   // underlying data source (can return nil)
}

// A FileMode represents a file's mode and permission bits.
// The bits have the same definition on all systems, so that
// information about files can be moved from one system
// to another portably. Not all bits apply to all systems.
```

It is a concrete type we need to use interface instead of this

2)

```
)

func ReadFileContents(rc io.ReadCloser, numBytes int) ([]byte, error) {
    defer rc.Close()
    data := make([]byte, numBytes)
    _, err := rc.Read(data)
    if err != nil {
        return nil, err
    }
    return data, nil
}
```

3)



```go
type mockReadCloser struct {
    expectedData []byte
    expectedErr  error
}

// allow test table to fill out expectedData and expectedErr field
func (mrc *mockReadCloser) Read(p []byte) (n int, err error) {
    copy(p, mrc.expectedData)
    return n: 0, mrc.expectedErr
}

// do nothing on close
func (mrc *mockReadCloser) Close() error { return nil }

func TestReadContents(t *testing.T) {
    errRead := errors.New( text: "uh oh")
    testTable := []struct {
        name         string
        readCloser   io.ReadCloser
        numBytes     int
        expectedData []byte
        expectedErr  error
    }{
        {
            name: "happy path",
            readCloser: &mockReadCloser{
                expectedData: []byte(`hello`),
                expectedErr:  nil,
            },
            numBytes:     5,
            expectedData: []byte(`hello`),
            expectedErr:  nil,
        },
        {
            name: "error-from-read",
            readCloser: &mockReadCloser{
                expectedData: nil,
                expectedErr:  errRead,
            },
            numBytes:     0,
            expectedData: nil,
            expectedErr:  errRead,
        },
```

4)

Interface Substitution

- “Accept interfaces, return structs”
- We used *io.ReadCloser*, but don’t be afraid to define your own interface

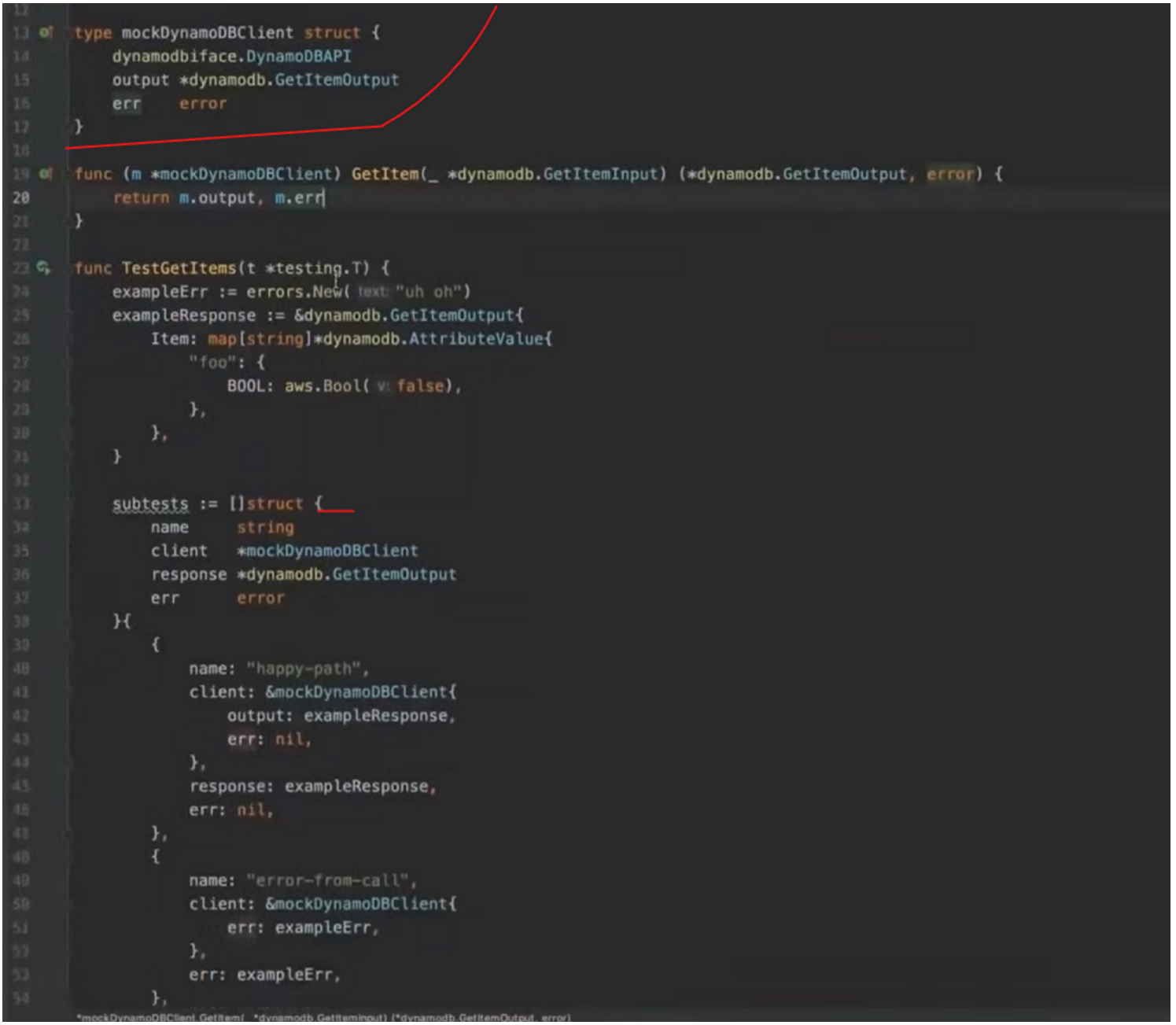
If I have a large Interface and I have to mock a small set of its methods

```go
func GetItems(client *dynamodb.DynamoDB) (*dynamodb.GetItemOutput, error) {
    return client.GetItem(&dynamodb.GetItemInput{
        AttributesToGet: []*string{aws.String( v: "bar")},
        Key: map[string]*dynamodb.AttributeValue{
            "foo": {
                BOOL: aws.Bool( v: true),
            },
        },
    })
}
```

```go
func GetItems(client dynamodbiface.DynamoDBAPI) (*dynamodb.GetItemOutput, error) {
    return client.GetItem(&dynamodb.GetItemInput{
        AttributesToGet: []*string{aws.String( v: "bar")},
        Key: map[string]*dynamodb.AttributeValue{
            "foo": {
                BOOL: aws.Bool( v: true),
            },
        },
    })
}
```

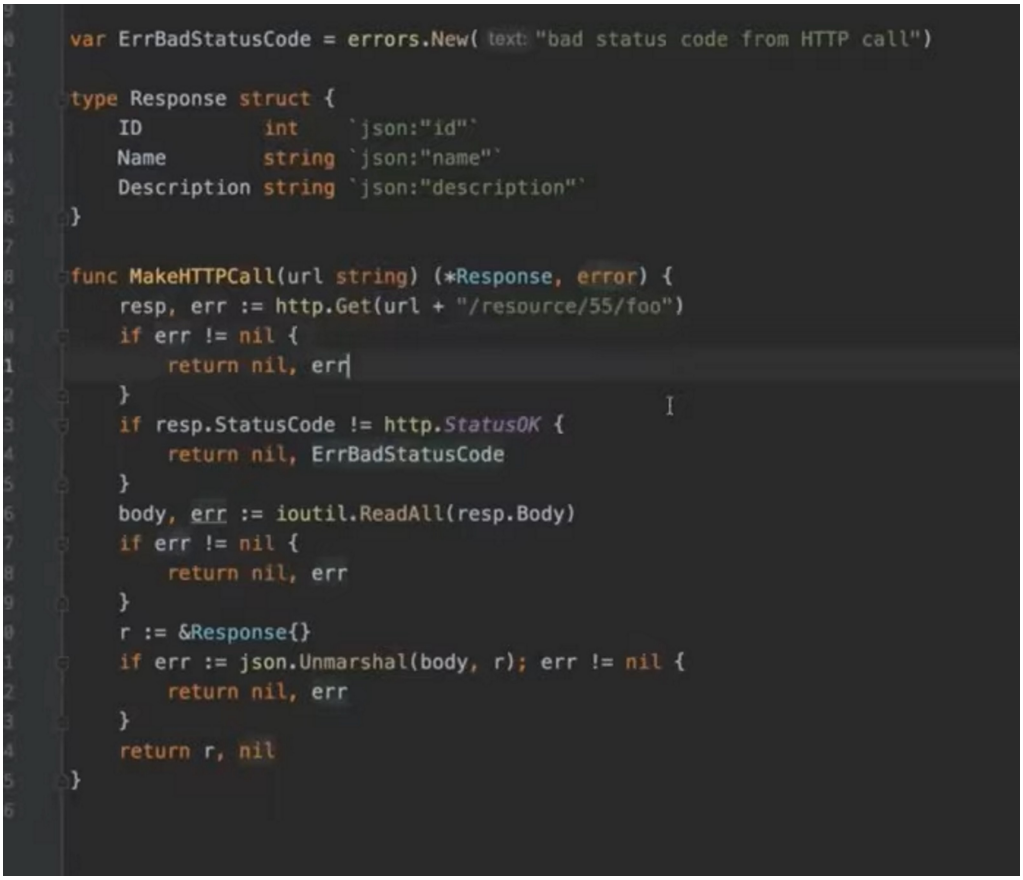
Dynamodb provides an interface for mocking

```go
    DescribeLimitsWithContext(aws.Context, *dynamodb.DescribeLimitsInput, ...request.Option) (*dynamodb.D
    DescribeLimitsRequest(*dynamodb.DescribeLimitsInput) (*request.Request, *dynamodb.DescribeLimitsOutput

    DescribeTable(*dynamodb.DescribeTableInput) (*dynamodb.DescribeTableOutput, error)
    DescribeTableWithContext(aws.Context, *dynamodb.DescribeTableInput, ...request.Option) (*dynamodb.Desc
    DescribeTableRequest(*dynamodb.DescribeTableInput) (*request.Request, *dynamodb.DescribeTableOutput)

    DescribeTableReplicaAutoScaling(*dynamodb.DescribeTableReplicaAutoScalingInput) (*dynamodb.DescribeTa
    DescribeTableReplicaAutoScalingWithContext(aws.Context, *dynamodb.DescribeTableReplicaAutoScalingInput
    DescribeTableReplicaAutoScalingRequest(*dynamodb.DescribeTableReplicaAutoScalingInput) (*request.Reque

    DescribeTimeToLive(*dynamodb.DescribeTimeToLiveInput) (*dynamodb.DescribeTimeToLiveOutput, error)
    DescribeTimeToLiveWithContext(aws.Context, *dynamodb.DescribeTimeToLiveInput, ...request.Option) (*dy
    DescribeTimeToLiveRequest(*dynamodb.DescribeTimeToLiveInput) (*request.Request, *dynamodb.DescribeTime

    GetItem(*dynamodb.GetItemInput) (*dynamodb.GetItemOutput, error)
    GetItemWithContext(aws.Context, *dynamodb.GetItemInput, ...request.Option) (*dynamodb.GetItemOutput,
    GetItemRequest(*dynamodb.GetItemInput) (*request.Request, *dynamodb.GetItemOutput)

    ListBackups(*dynamodb.ListBackupsInput) (*dynamodb.ListBackupsOutput, error)
    ListBackupsWithContext(aws.Context, *dynamodb.ListBackupsInput, ...request.Option) (*dynamodb.ListBack
    ListBackupsRequest(*dynamodb.ListBackupsInput) (*request.Request, *dynamodb.ListBackupsOutput)

    ListContributorInsights(*dynamodb.ListContributorInsightsInput) (*dynamodb.ListContributorInsightsOut
    ListContributorInsightsWithContext(aws.Context, *dynamodb.ListContributorInsightsInput, ...request.Op
    ListContributorInsightsRequest(*dynamodb.ListContributorInsightsInput) (*request.Request, *dynamodb.L
```

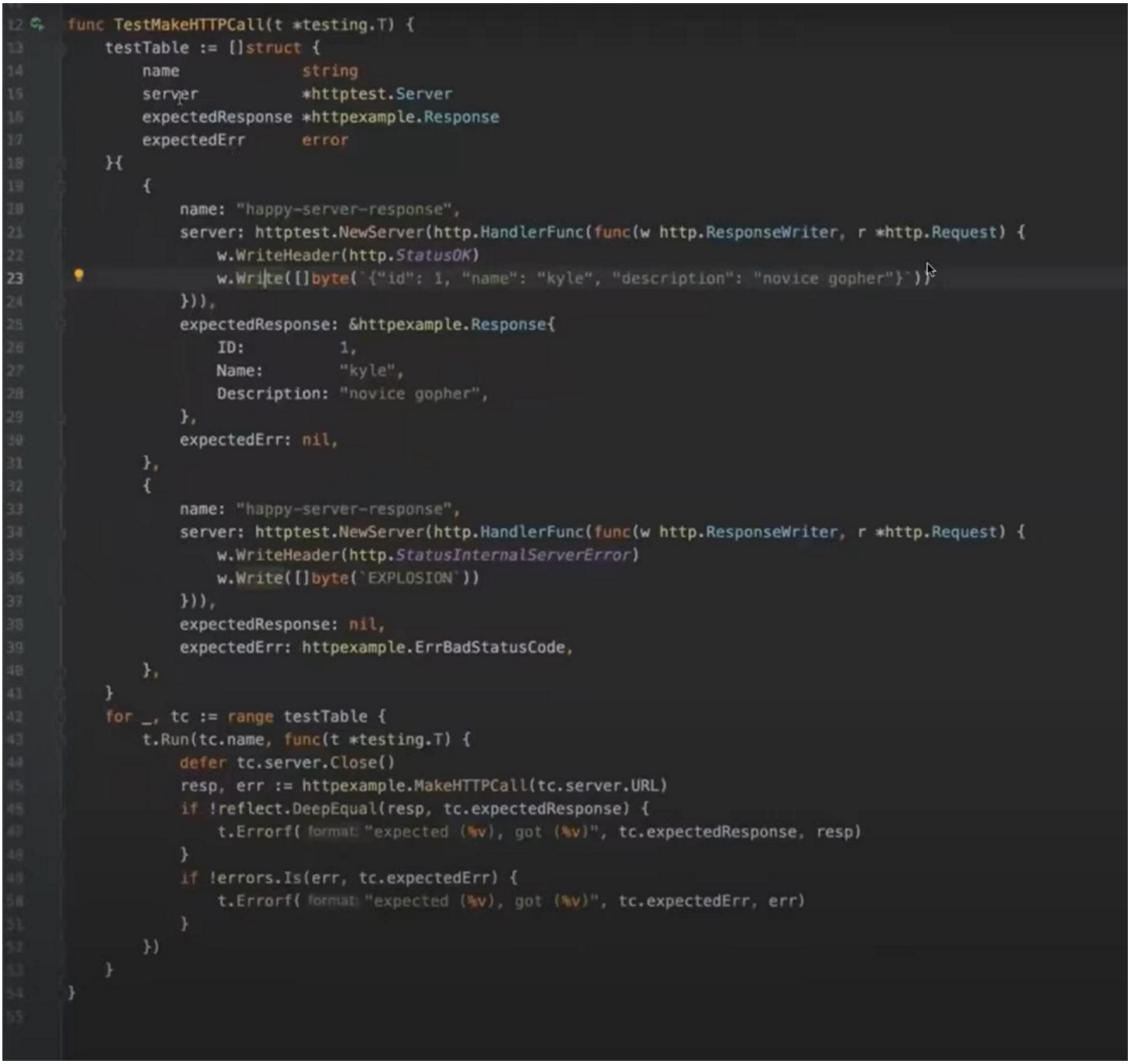


```go
type mockDynamoDBClient struct {
    dynamodbiface.DynamoDBAPI
    output *dynamodb.GetItemOutput
    err    error
}

func (m *mockDynamoDBClient) GetItem(_ *dynamodb.GetItemInput) (*dynamodb.GetItemOutput, error) {
    return m.output, m.err
}

func TestGetItems(t *testing.T) {
    exampleErr := errors.New( text: "uh oh")
    exampleResponse := &dynamodb.GetItemOutput{
        Item: map[string]*dynamodb.AttributeValue{
            "foo": {
                BOOL: aws.Bool( v: false),
            },
        },
    }

    subtests := []struct {
        name     string
        client   *mockDynamoDBClient
        response *dynamodb.GetItemOutput
        err      error
    }{
        {
            name: "happy-path",
            client: &mockDynamoDBClient{
                output: exampleResponse,
                err: nil,
            },
            response: exampleResponse,
            err: nil,
        },
        {
            name: "error-from-call",
            client: &mockDynamoDBClient{
                err: exampleErr,
            },
            err: exampleErr,
        },
```

I need to mock an Http call



```go
var ErrBadStatusCode = errors.New( text: "bad status code from HTTP call")

type Response struct {
    ID          int    `json:"id"`
    Name        string `json:"name"`
    Description string `json:"description"`
}

func MakeHTTPCall(url string) (*Response, error) {
    resp, err := http.Get(url + "/resource/55/foo")
    if err != nil {
        return nil, err
    }
    if resp.StatusCode != http.StatusOK {
        return nil, ErrBadStatusCode
    }
    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return nil, err
    }
    r := &Response{}
    if err := json.Unmarshal(body, r); err != nil {
        return nil, err
    }
    return r, nil
}
```



```
func TestMakeHTTPCall(t *testing.T) {
    testTable := []struct {
        name             string
        server           *httptest.Server
        expectedResponse *httpexample.Response
        expectedErr      error
    }{
        {
            name: "happy-server-response",
            server: httptest.NewServer(http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
                w.WriteHeader(http.StatusOK)
                w.Write([]byte(`{"id": 1, "name": "kyle", "description": "novice gopher"}`))
            })),
            expectedResponse: &httpexample.Response{
                ID:          1,
                Name:        "kyle",
                Description: "novice gopher",
            },
            expectedErr: nil,
        },
        {
            name: "happy-server-response",
            server: httptest.NewServer(http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
                w.WriteHeader(http.StatusInternalServerError)
                w.Write([]byte(`EXPLOSION`))
            })),
            expectedResponse: nil,
            expectedErr: httpexample.ErrBadStatusCode,
        },
    }
    for _, tc := range testTable {
        t.Run(tc.name, func(t *testing.T) {
            defer tc.server.Close()
            resp, err := httpexample.MakeHTTPCall(tc.server.URL)
            if !reflect.DeepEqual(resp, tc.expectedResponse) {
                t.Errorf( format "expected (%v), got (%v)", tc.expectedResponse, resp)
            }
            if !errors.Is(err, tc.expectedErr) {
                t.Errorf( format "expected (%v), got (%v)", tc.expectedErr, err)
            }
        })
    }
}
```

net/http/httptest

We made one test server per subtest, but you could make one server with many routes, just a like a normal server!

`