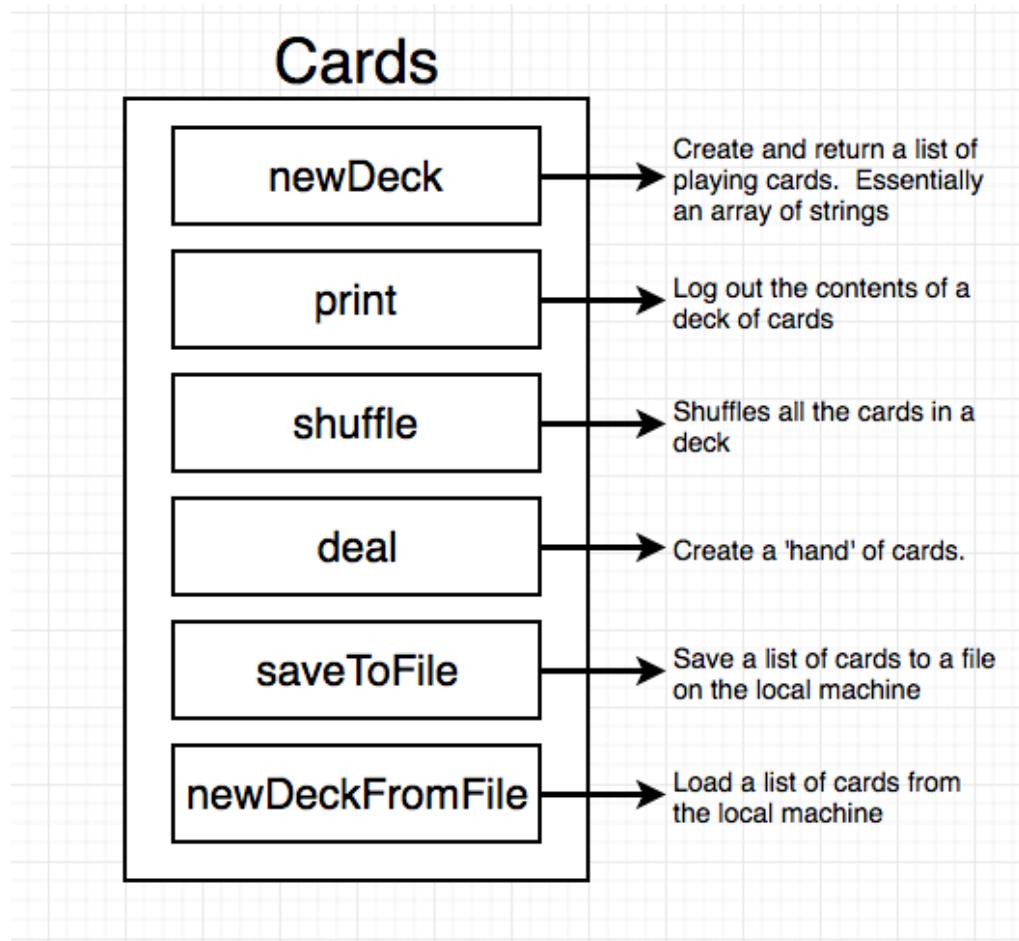


# 3 - Deeper Into Go

## Project Overview



## New Project Folder

src/cards

## Variable Declarations

src/cards/main.go

```
1 func main() {  
2     var card = "Ace of Spades"  
3     fmt.Println(card)  
4 }
```

```
var card string = "Ace of Spades"
```

We're about to  
create a new  
variable

The name of the  
variable will be  
'card'

Only a "string" will  
ever be assigned  
to this variable

Assign the value  
"Ace of Spades"  
to this variable

### Dynamic Types

Javascript

Ruby

Python

### Static Types

C++

Java

Go

## Basic Go Types

Type	Example	
bool	true	false
string	"Hi!"	"Hows it going?"
int	0	-10000 99999
float64	10.000001	0.00009 -100.003

```
1 func main() {  
2     // var card = "Ace of Spades"  
3     card := "Ace of Spades"  
4     fmt.Println(card)  
5 }
```

## Functions and Return Types

```
1 func main() {  
2     card := newCard()  
3     fmt.Println(card)  
4 }  
  
1 func newCard() string {  
2     return "Five of Diamonds"  
3 }
```

Define a function called 'newCard'

When executed, this function will return a value of type 'string'

func newCard() string {  
}  
}

## Slices and For Loops

Array

*Fixed length list of things*

Slice

*An array that can grow or shrink*

## Slices

"Five of Spades"	"Three of Diamonds"	"Five of Diamonds"
------------------	---------------------	--------------------

*Every element in a slice must  
be of same type*

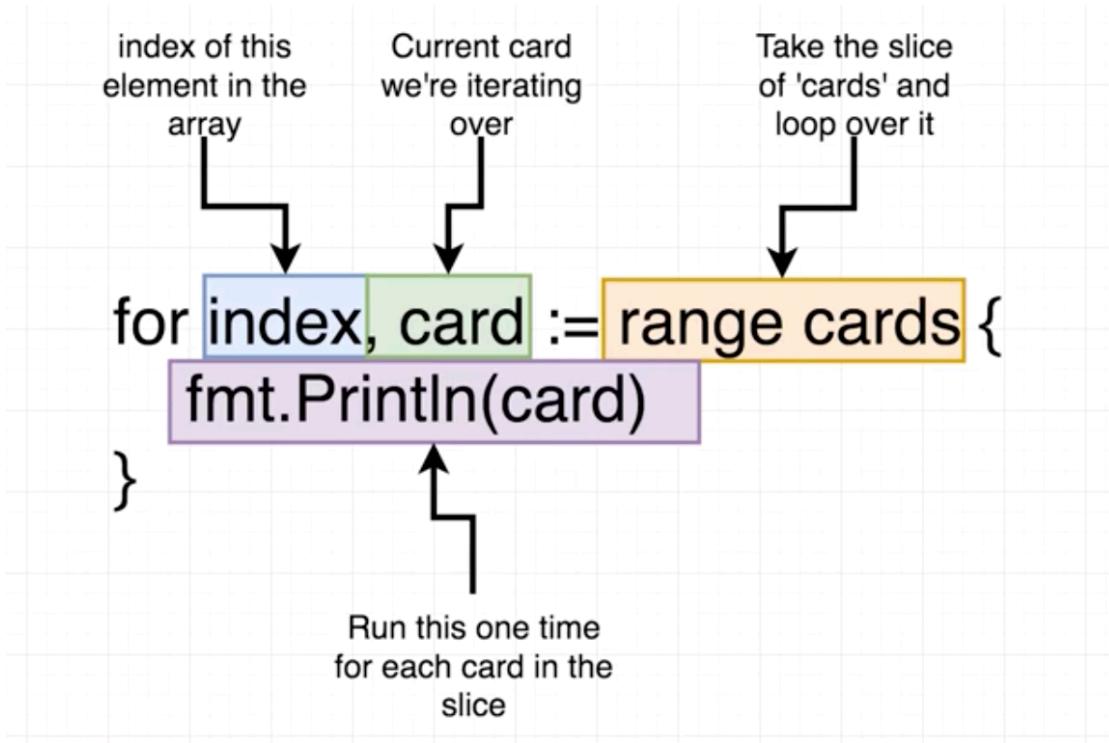
"Five of Spades"	55525205	"Five of Diamonds"
------------------	----------	--------------------

### Slice example and Add new Item

```
1 func main() {  
2     cards := []string{"Ace of Diamonds", newCard()}  
3     // Add new item  
4     cards = append(cards, "Six of Spades")  
5     fmt.Println(cards)  
6 }
```

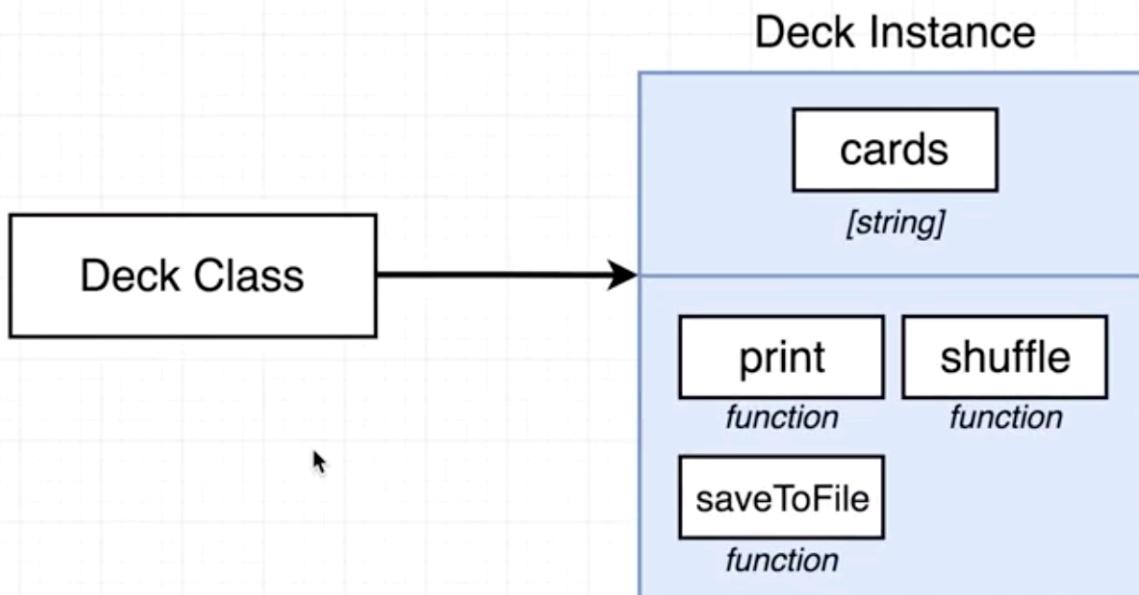
### Iterate slice

```
1 func main() {  
2     cards := []string{"Ace of Diamonds", newCard(), "Six of Spades"}  
3  
4     // Iterate slice  
5     for i, card := range cards {  
6         fmt.Println(i, card)  
7     }  
8 }
```

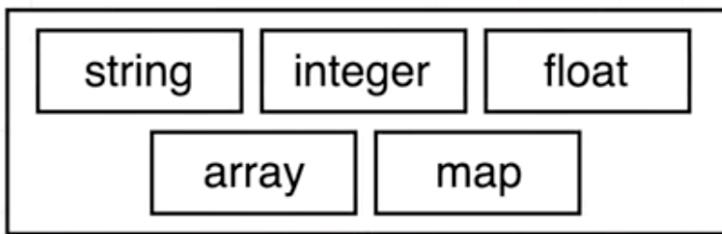


## OO Approach vs Go Approach

### Cards - OO Approach



## Base Go Types



We want to "extend" a base type and add some extra functionality to it

type deck []string

Tell Go we want to create an array of strings and add a bunch of functions specifically made to work with it

Functions with 'deck'  
as a 'receiver'

A function with a receiver is like a "method" - a function that belongs to an "instance"

## 'cards' folder



Code to create  
and manipulate  
a deck

Code that  
describes what  
a deck is and  
how it works

Code to  
automatically  
test the deck

## Custom Type Declarations

/src/cards/deck.go

```
1 package main
2
3 import "fmt"
4
5 // Create a new type of 'deck'
6 // which is a slice of strings
```

```
7 type deck []string
8
9 func (d deck) print() {
10    for index, card := range d {
11        fmt.Println(index, card)
12    }
13 }
14
```

/src/cards/main.go

```
1 package main
2
3 func main() {
4     cards := deck{"Ace of Diamonds", newCard()}
5     cards = append(cards, "Six of Spades")
6
7     cards.print()
8 }
9
10 func newCard() string {
11     return "Five of Diamonds"
12 }
```

```
1 $ go run main.go deck.go
```

## Receiver Functions



```
func (d deck) print() {
```

Any variable of type "deck"  
now gets access to the  
"print" method

The actual copy of the deck we're working with is available in the function as a variable called 'd'

Every variable of type 'deck' can call this function on itself

```
func(d deck) print() {  
    for i, card := range d {  
        fmt.Println(i, card)  
    }  
}
```

## Creating a New Deck

```
cards = deck{}
```

```
cardSuits := []string{"Spades", "Hearts", "Diamonds"}
```

```
cardValues := []string{"Ace", "Two", "Three"}
```

```
for each suit in cardSuits
```

```
    for each value in cardValues
```

```
        Add a new card of 'value of suit' to the 'cards' deck
```

/src/cards/deck.go

```
1 // Create a new type of 'deck'  
2 // which is a slice of strings  
3 type deck []string  
4
```

```

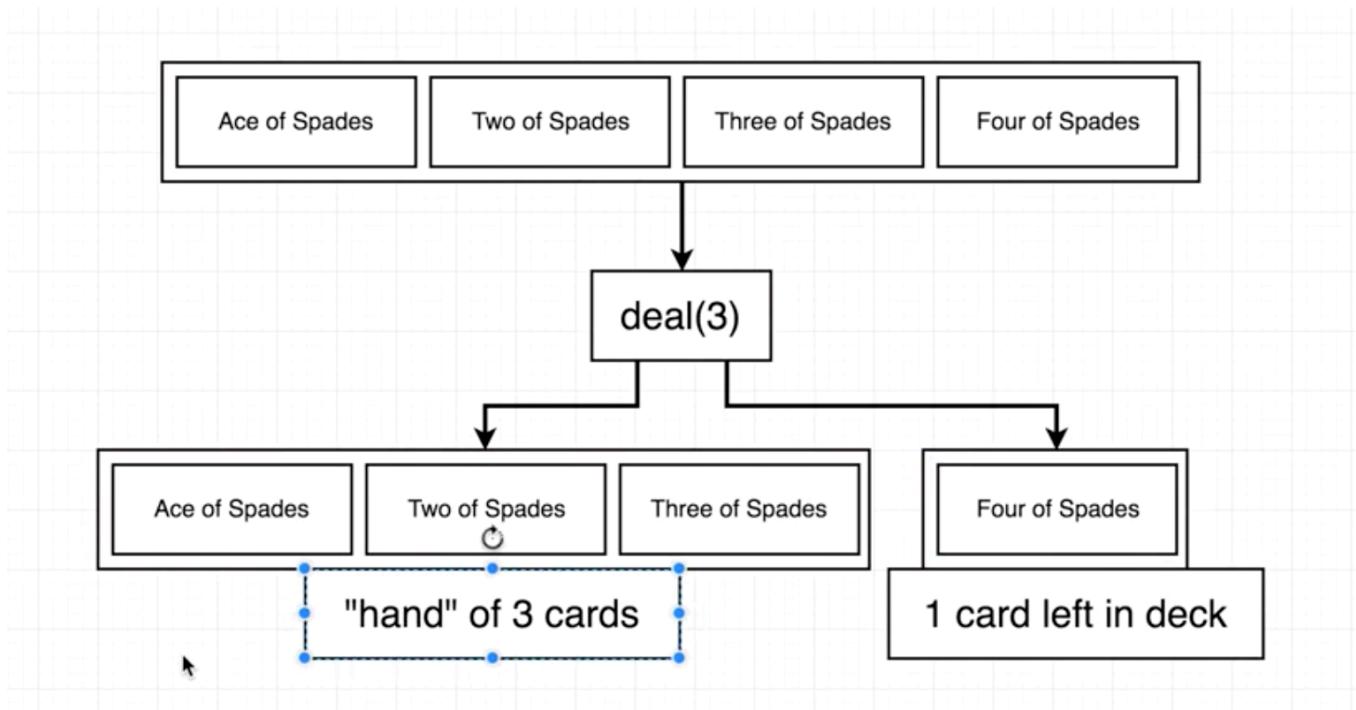
5 func newDeck() deck {
6     cards := deck{}
7
8     cardSuits := []string{"Spades", "Diamonds", "Hearts", "Clubs"}
9     cardValues := []string{"Ace", "Two", "Three", "Four"}
10
11    for _, suit := range cardSuits {
12        for _, value := range cardValues {
13            cards = append(cards, suit+" of "+value)
14        }
15    }
16
17    return cards
18
19 }
20
21 func (d deck) print() {
22     for index, card := range d {
23         fmt.Println(index, card)
24     }
25 }
```

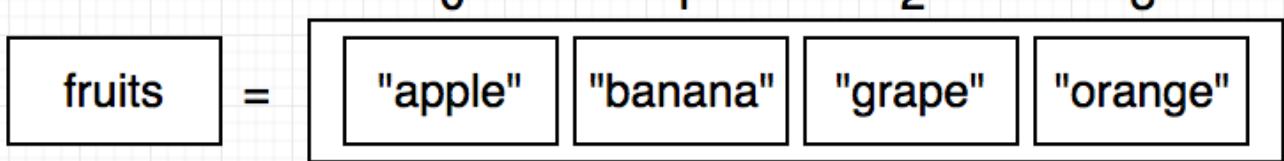
/src/cards/main.go

```

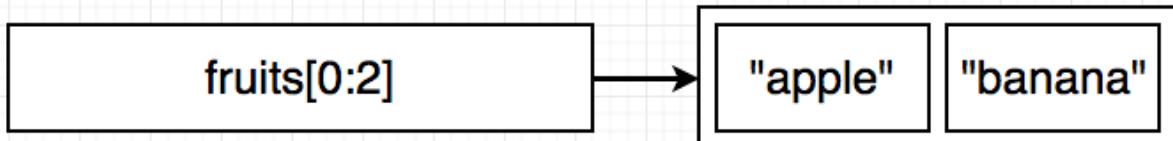
1 func main() {
2     cards := newDeck()
3     cards.print()
4 }
```

## Slice Range Syntax





`fruits[startIndexInclusive : upToNotInclusive]`



## Multipole Return Values

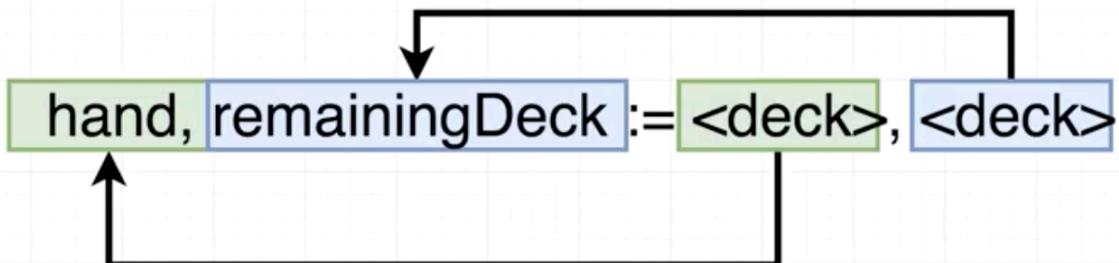
/src/cards/deck.go

```
1 func deal(d deck, handSize int) (deck, deck) {
2     return d[:handSize], d[handSize:]
3 }
```

/src/cards/main.go

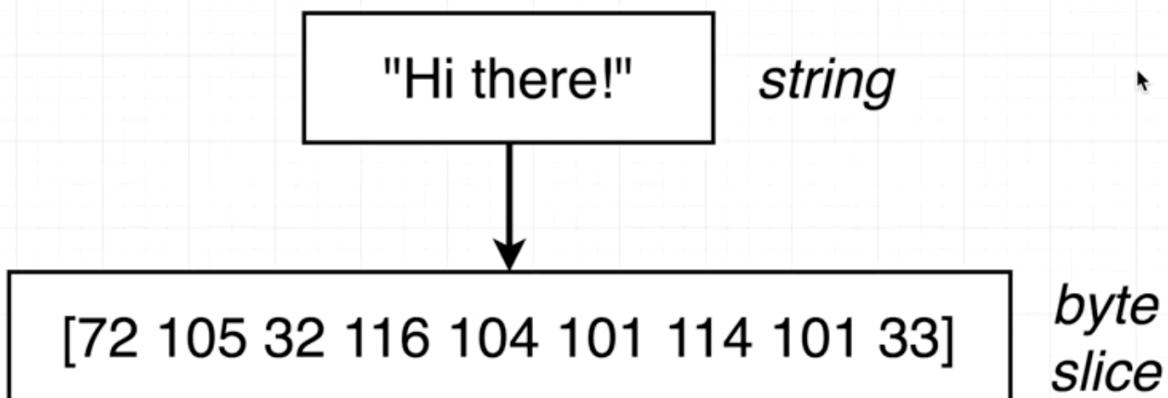
```
1 func main() {
2     cards := newDeck()
3     hand, remainingDeck := deal(cards, 5)
4
5     hand.print()
6     remainingDeck.print()
7 }
```

hand, remainingDeck = deal(cards, 5)



## Byte Slices

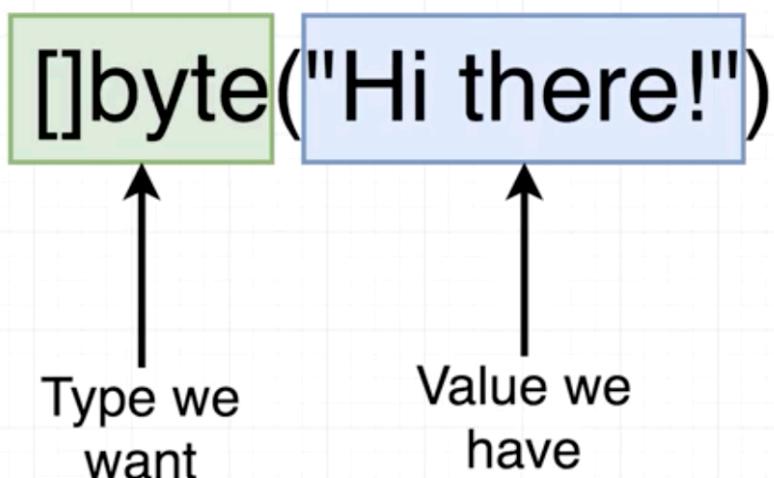
<https://golang.org/pkg/io/ioutil/>



asciitable.com

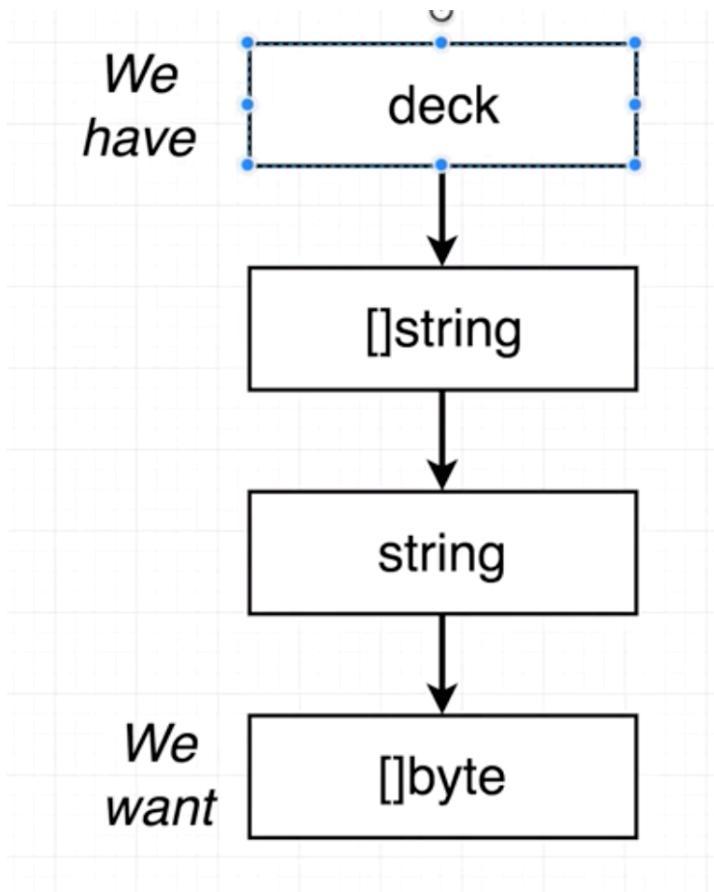
## Deck to String

### Type Conversion



/src/cards/main.go

```
1 func main() {  
2     greeting := "Hi there! "  
3     fmt.Println([]byte(greeting))  
4 }
```



## Joining a Slice of Strings

/src/cards/deck.go

```

1 func (d deck) toString() string {
2     return strings.Join([]string(d), ",")
3 }

```

<https://golang.org/pkg/strings/#Join>

/src/cards/main.go

```

1 func main() {
2     cards := newDeck()
3     fmt.Println(cards.toString())
4 }

```

## Saving Data to the Hard Drive

/src/cards/deck.go

```

1 func (d deck) saveToFile(filename string) error {
2     return ioutil.WriteFile(filename, []byte(d.toString()), 0666)
3 }

```

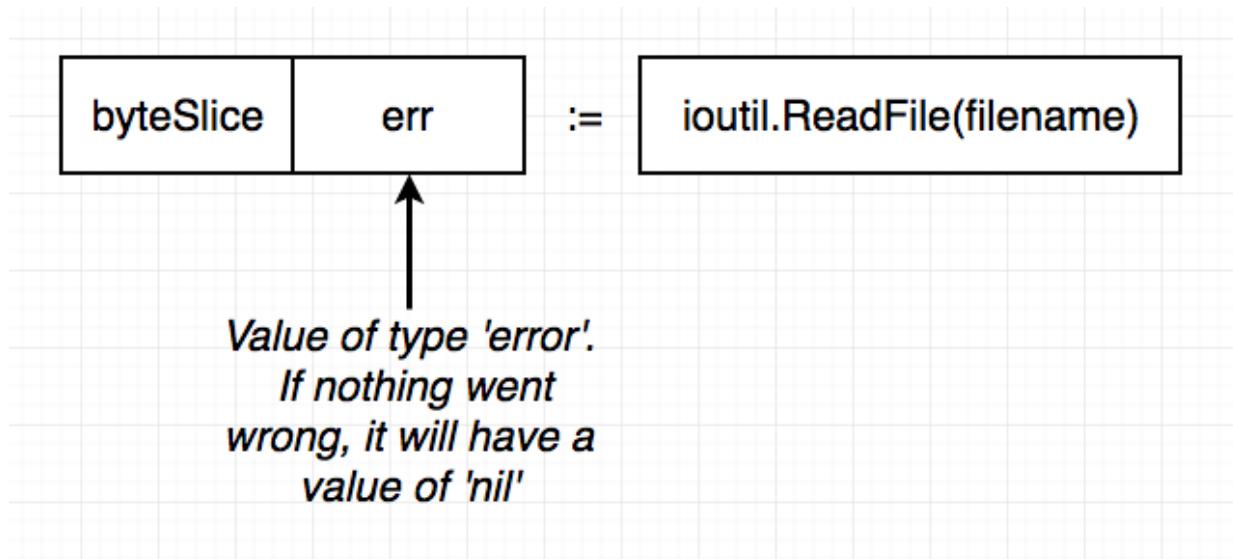
/src/cards/main.go

```

1 func main() {
2     cards := newDeck()
3     cards.saveToFile("my_cards")
4 }

```

## Reading From the Hard Drive



<https://golang.org/pkg/io/ioutil/#ReadFile>

/src/cards/deck.go

```

1 func newDeckFromFile(filename string) deck {
2     byteSlice, err := ioutil.ReadFile(filename)
3     if err != nil {
4         // Option #1 - log the error and return a call to newDeck()
5         // Option #2 - log the error and entirely quit the program
6         fmt.Println("Error: ", err)
7         os.Exit(1)
8     }
9
10    stringSlice := strings.Split(string(byteSlice), ",") // Ace of Spades,Two of
11        Spades,Three of Spades,....
12    return deck(stringSlice)
13 }

```

/src/cards/main.go

```

1 func main() {
2     cards := newDeckFromFile("my_cards")
3     cards.print()
4 }

```

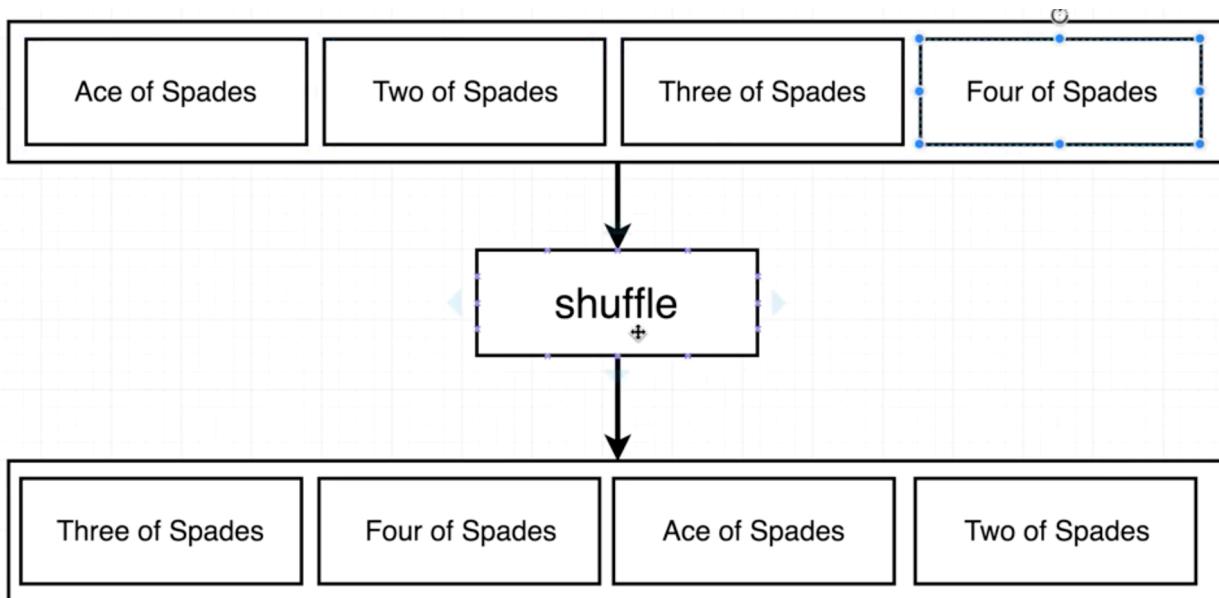
/src/cards/main.go (error)

```

1 func main() {
2     cards := newDeckFromFile("my")
3     cards.print()
4 }

```

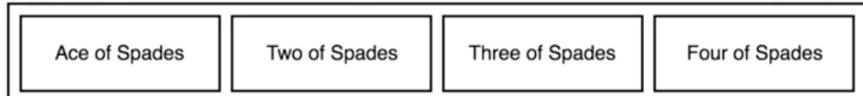
## Shuffling a Deck



for each index, card in cards

    Generate a random number between 0 and len(cards) - 1

    Swap the current card and the card at cards[randomNumber]



<https://golang.org/pkg/math/rand/#Intn>

/src/cards/deck.go

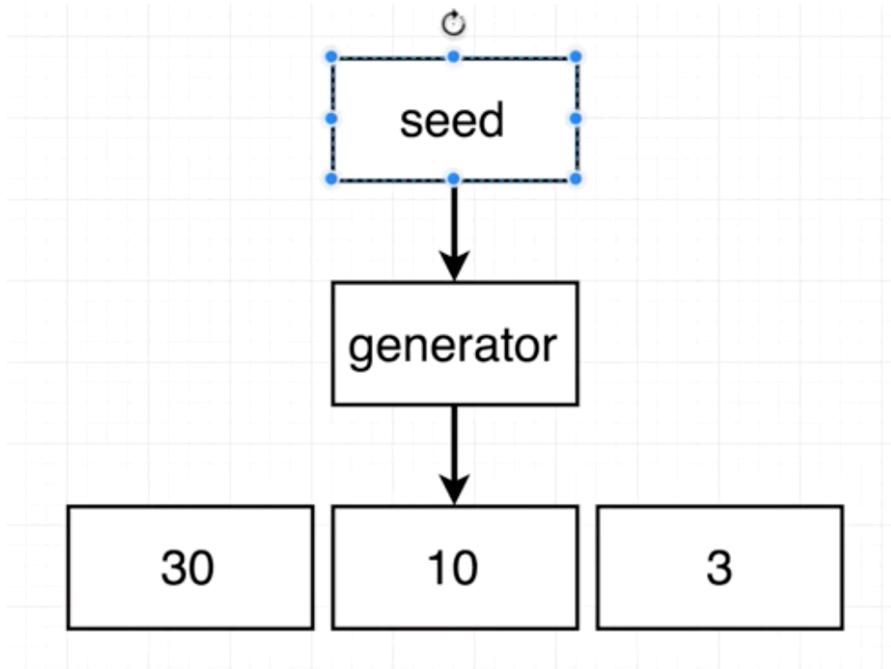
```
1 func (d deck) shuffle() {
2     for index := range d {
3         newPosition := rand.Intn(len(d) - 1)
4         // swap item in index and in newPosition
5         d[index], d[newPosition] = d[newPosition], d[index]
6     }
7 }
```

/src/cards/main.go

```
1 func main() {
2     cards := newDeck()
3     cards.shuffle()
4     cards.print()
```

```
5 }
```

## Random Number Generation



<https://golang.org/pkg/math/rand/#Rand>

Better Approach to generate random number to change numbers every time

/src/cards/deck.go

```
1 func (d deck) shuffle() {
2     source := rand.NewSource(time.Now().UnixNano())
3     r := rand.New(source)
4
5     for i := range d {
6         newPosition := r.Intn(len(d) - 1)
7         d[i], d[newPosition] = d[newPosition], d[i]
8     }
9 }
```

## Testing With Go

Go testing is not RSpec, mocha, jasmine, selenium, etc!

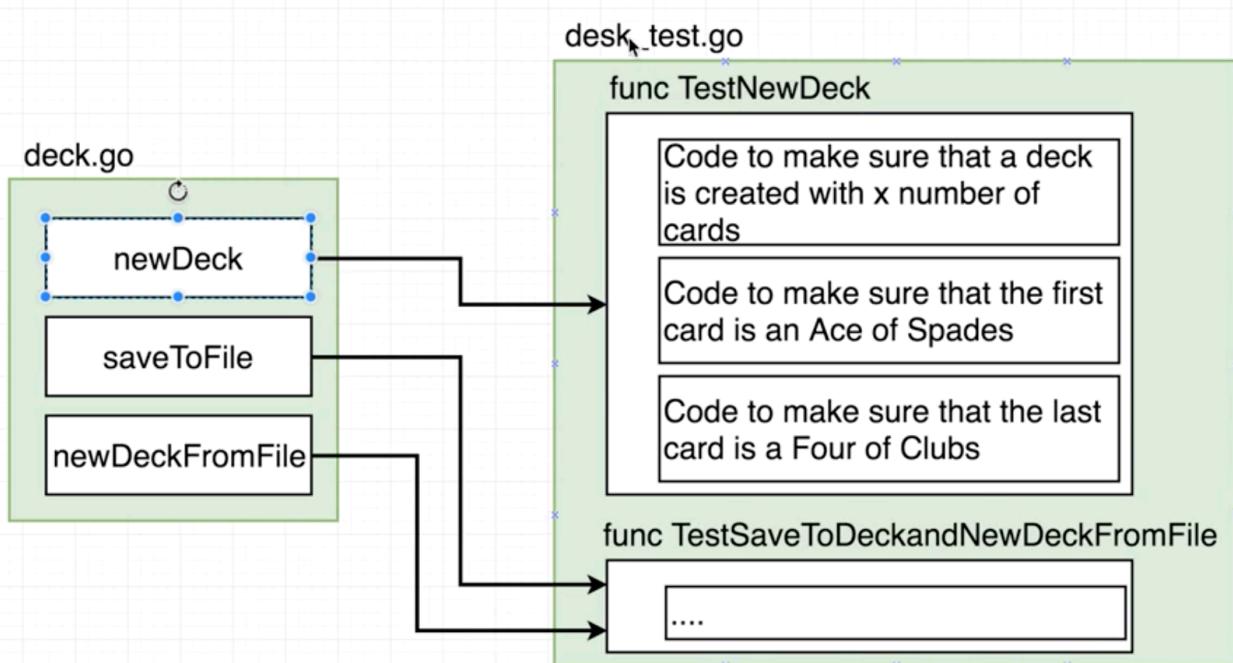
To make a test, create a new file ending in `_test.go`

deck\_test.go

To run all tests in a package, run the command...

go test

## Writing Useful Tests



To make a test, create a new file ending in `_test.go`

/src/cards/deck\_test.go

```
1 func TestNewDeck(t *testing.T) {
2     d := newDeck()
3
4     if len(d) != 16 {
5         t.Errorf("Expected deck length of 16, but got %v", len(d))
6     }
7
8     if d[0] != "Ace of Spades" {
```

```

9      t.Errorf("Expected first card of Ace of Spades, but got %v", d[0])
10 }
11
12 if d[len(d)-1] != "Four of Cubes" {
13     t.Errorf("Expected last card of Four of Cubes, but got %v", d[len(d)-1])
14 }
15 }
```

## Testing File IO

```

1 func TestSaveToDeckAndNewDeckFromFile(t *testing.T) {
2     os.Remove("_decktesting")
3
4     deck := newDeck()
5     deck.saveToFile("_decktesting")
6
7     loadedDeck := newDeckFromFile("_decktesting")
8
9     if len(loadedDeck) != 16 {
10         t.Errorf("Expected 16 cards in deck, got %v", len(loadedDeck))
11     }
12
13     os.Remove("_decktesting")
14 }
```

## Assignment 1: Even and Odd

```

1 func main() {
2     numbers := []int{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
3
4     for number := range numbers {
5         if number%2 == 0 {
6             fmt.Println(number, " is even")
7         } else {
8             fmt.Println(number, " is odd")
9         }
10    }
11 }
```

