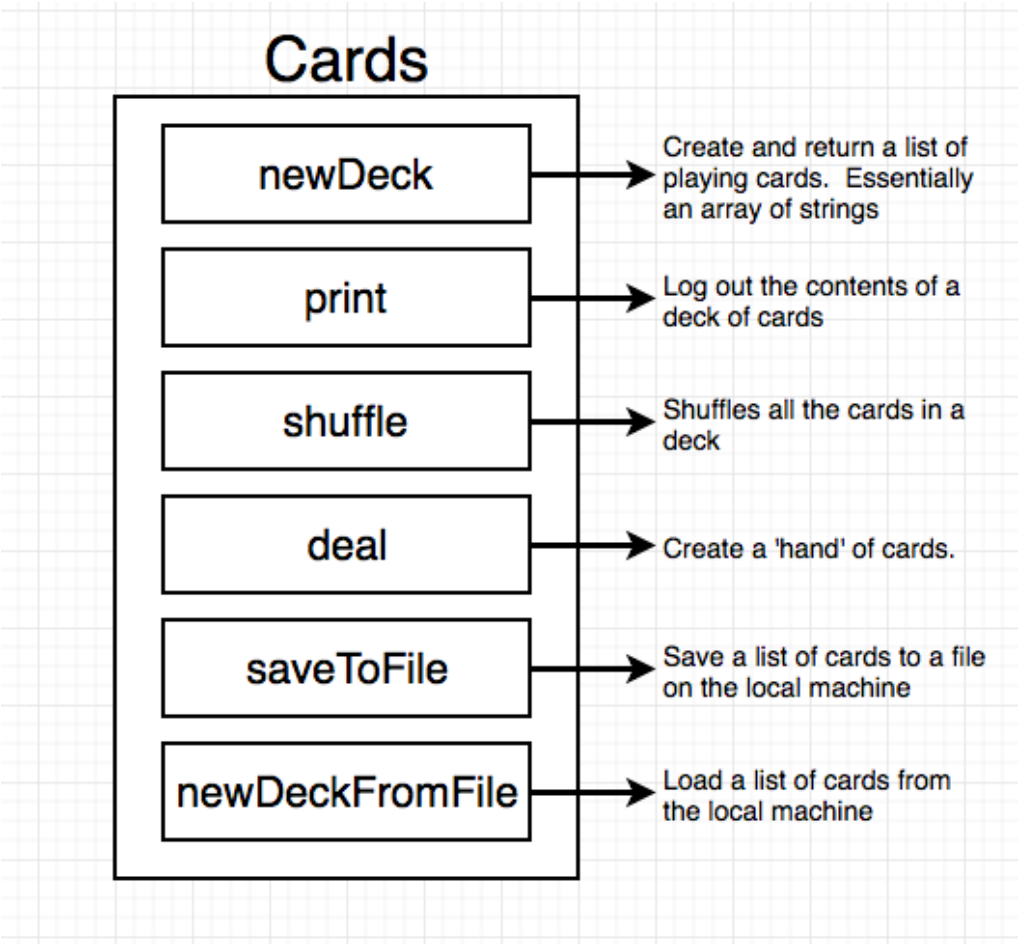
3 - Deeper Into Go

Project Overview



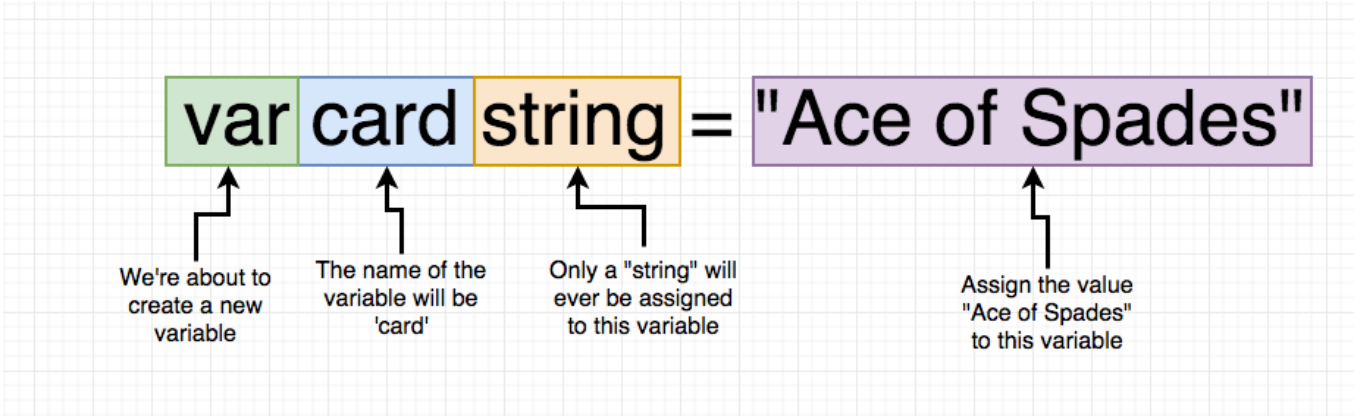
New Project Folder

src/cards

Variable Declarations

src/cards/main.go

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



Dynamic Types	Static Types
Javascript	C++
Ruby	Java
Python	Go

Basic Go Types

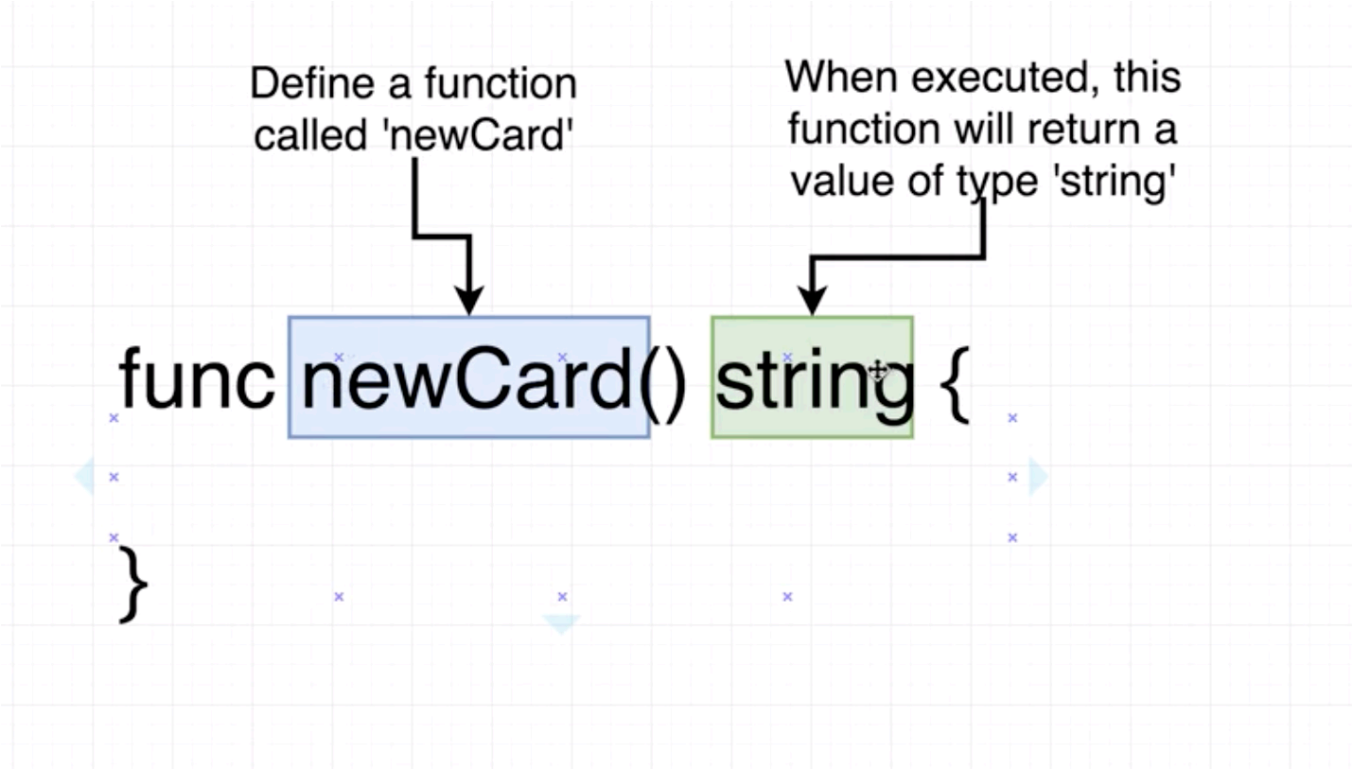
Type	Example
bool	true, false
string	"Hi!", "Hows it going?"
int	0, -10000, 99999
float64	10.000001, 0.00009, -100.003

```go
func main() {
    // var card = "Ace of Spades"
    card := "Ace of Spades"
    fmt.Println(card)
}
```

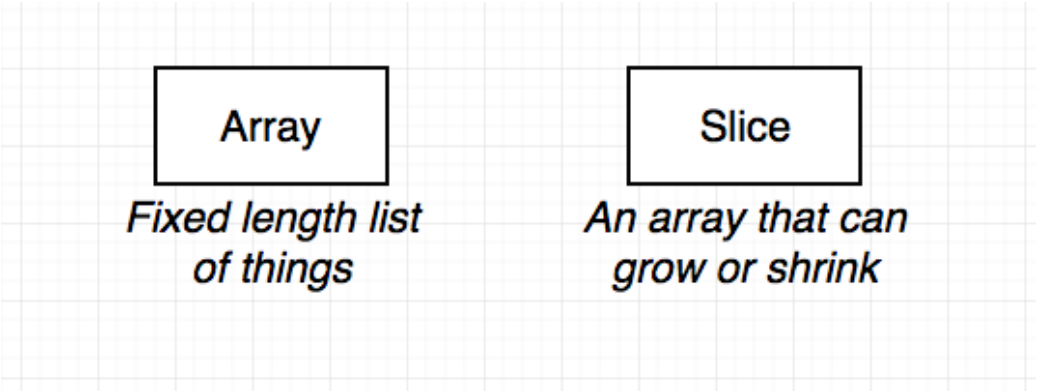
Functions and Return Types

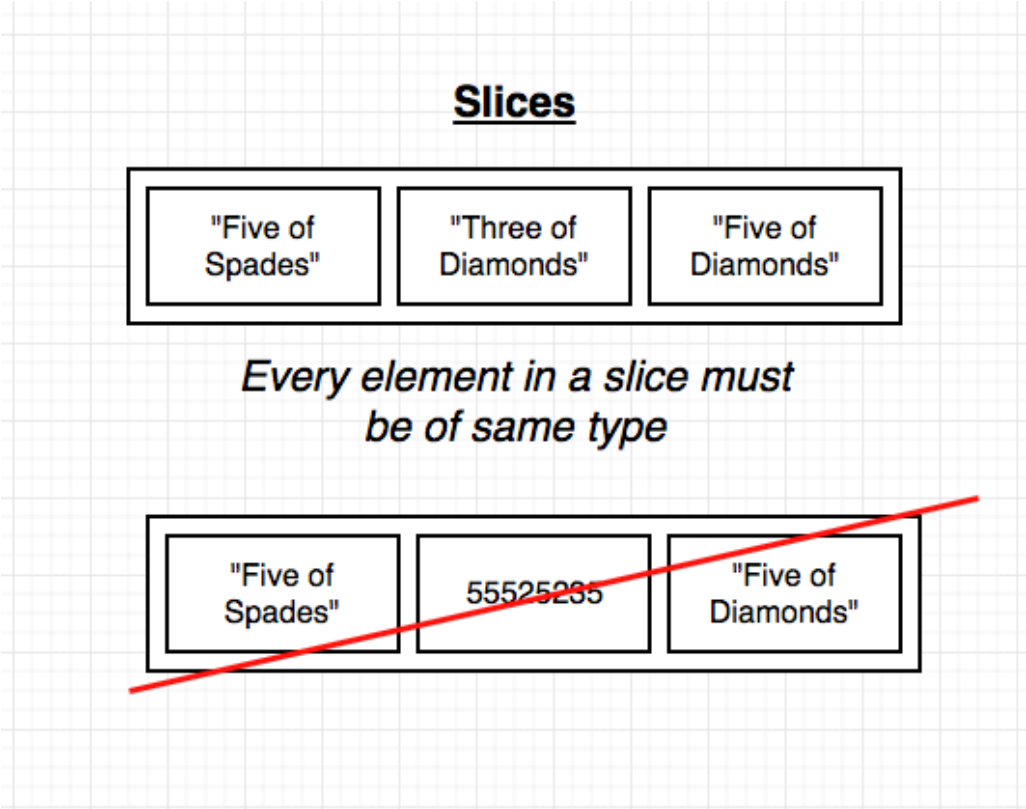
```go
func main() {
    card := newCard()
    fmt.Println(card)
}
```

```go
func newCard() string {
    return "Five of Diamonds"
}
```



Slices and For Loops



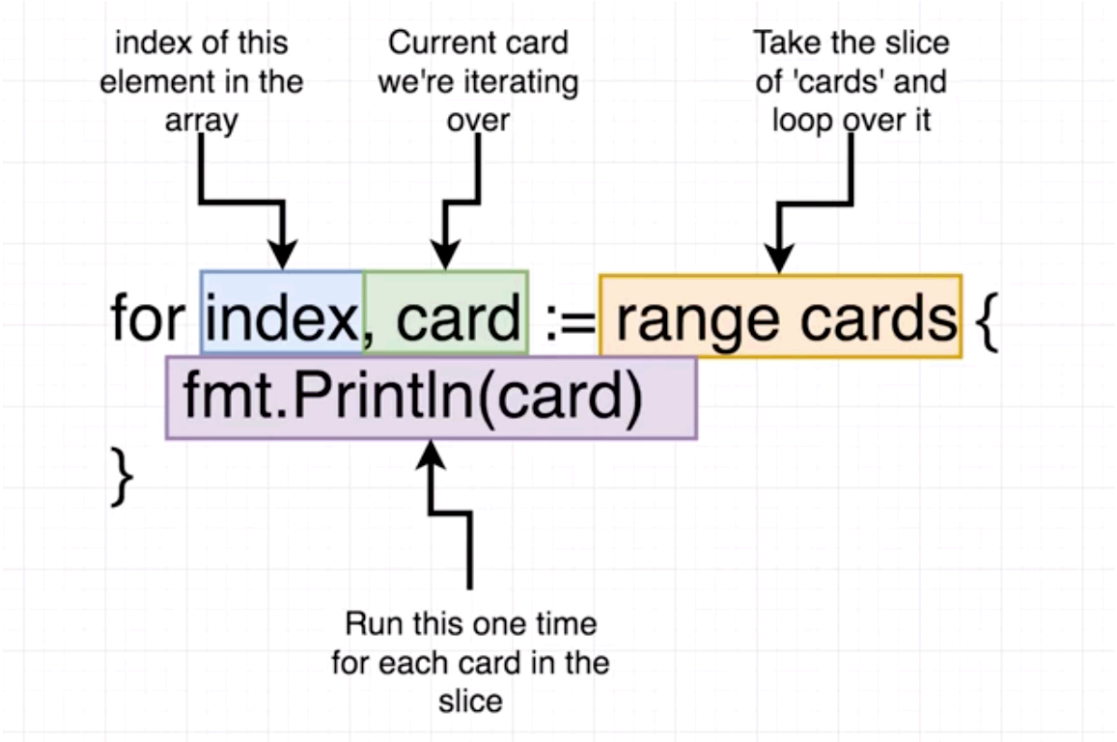


Slice example and Add new Item

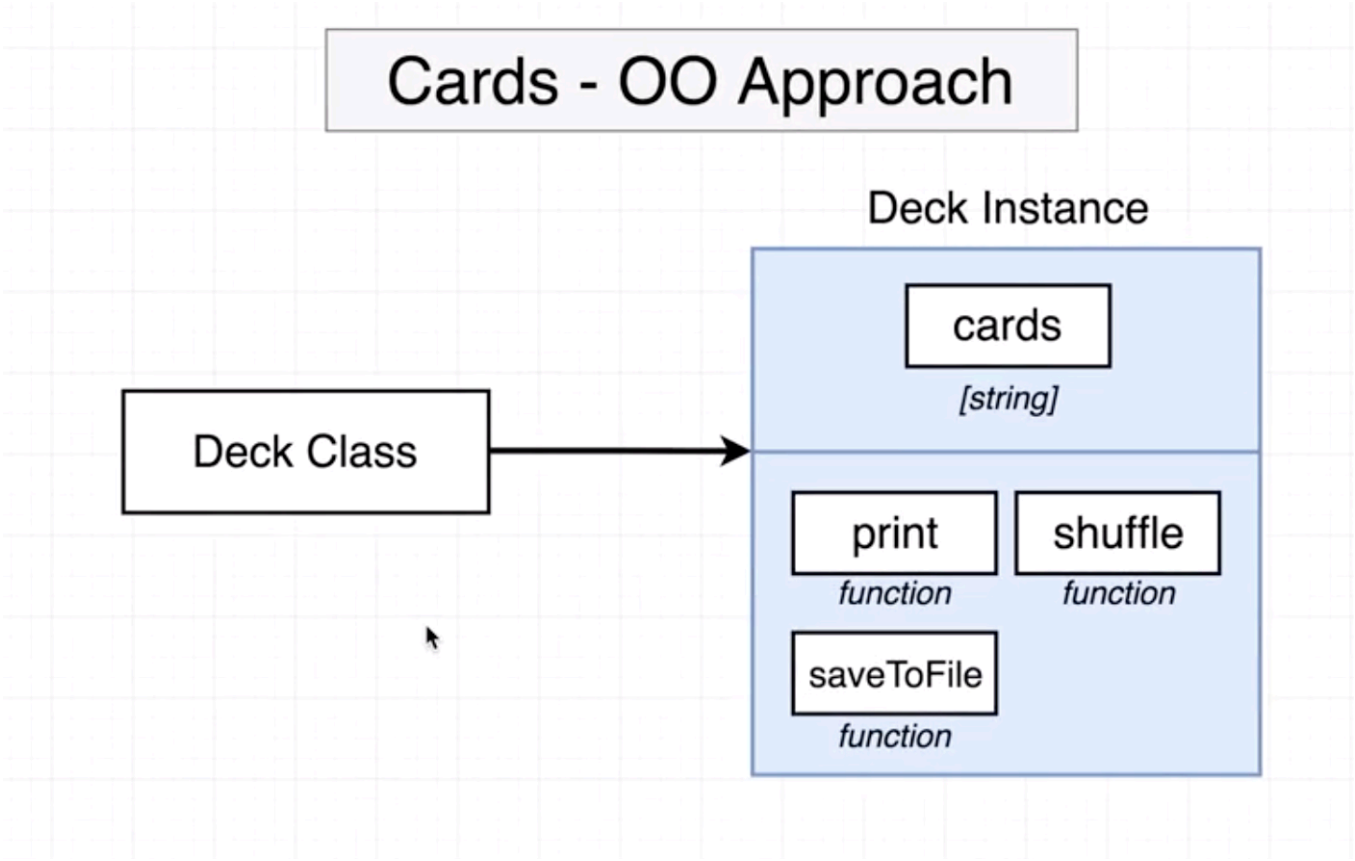
```
func main() {
    cards := []string{"Ace of Diamonds", newCard()}
    // Add new item
    cards = append(cards, "Six of Spades")
    fmt.Println(cards)
}
```

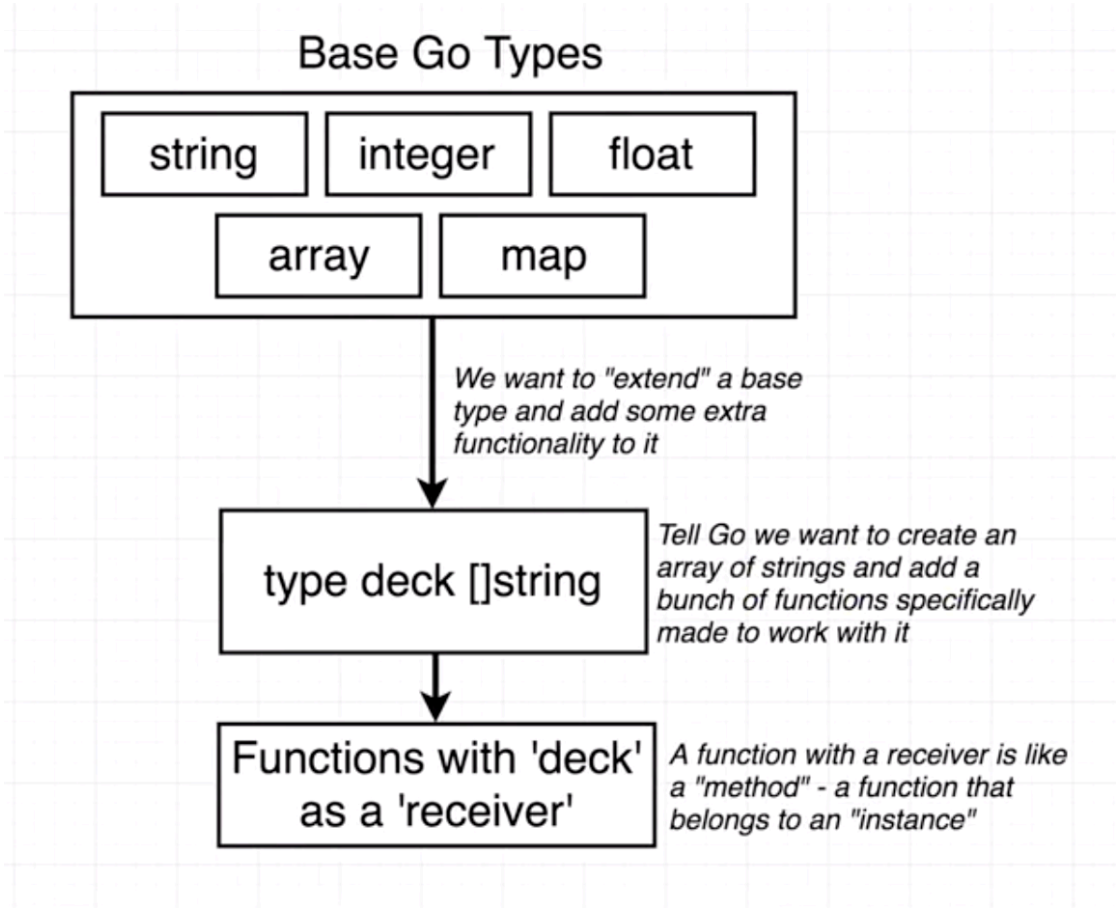
Iterate slice

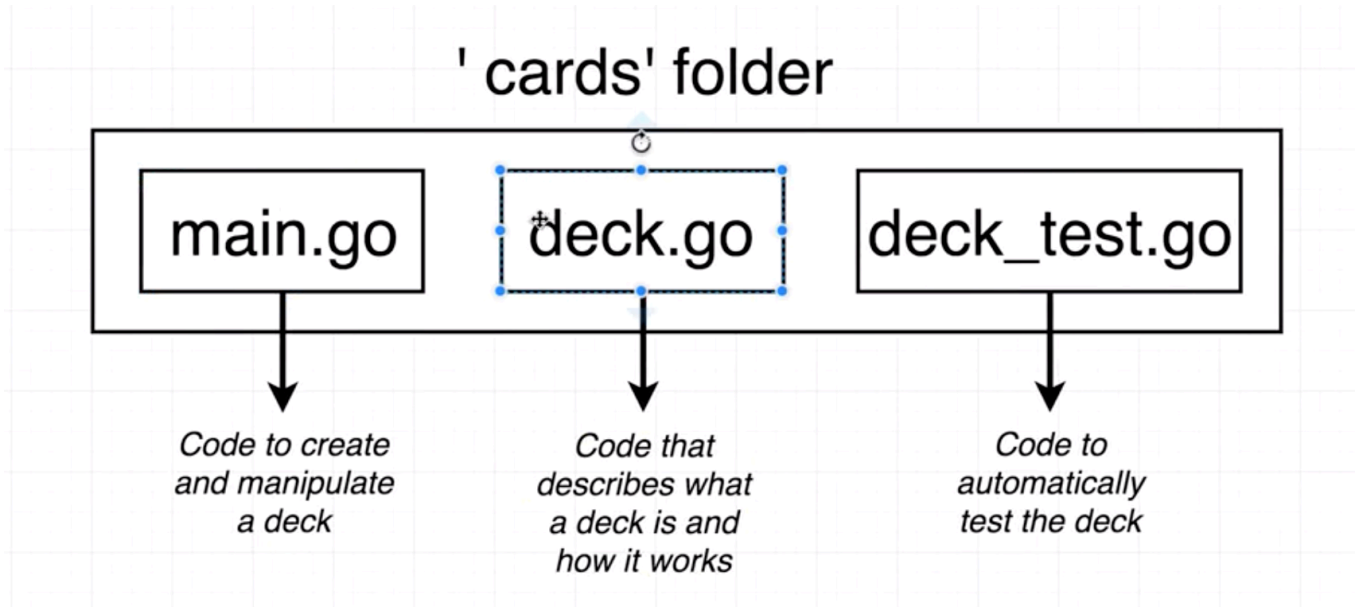
```
func main() {
    cards := []string{"Ace of Diamonds", newCard(), "Six of Spades"}

    // Iterate slice
    for i, card := range cards {
        fmt.Println(i, card)
    }
}
```



OO Approach vs Go Approach







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
```

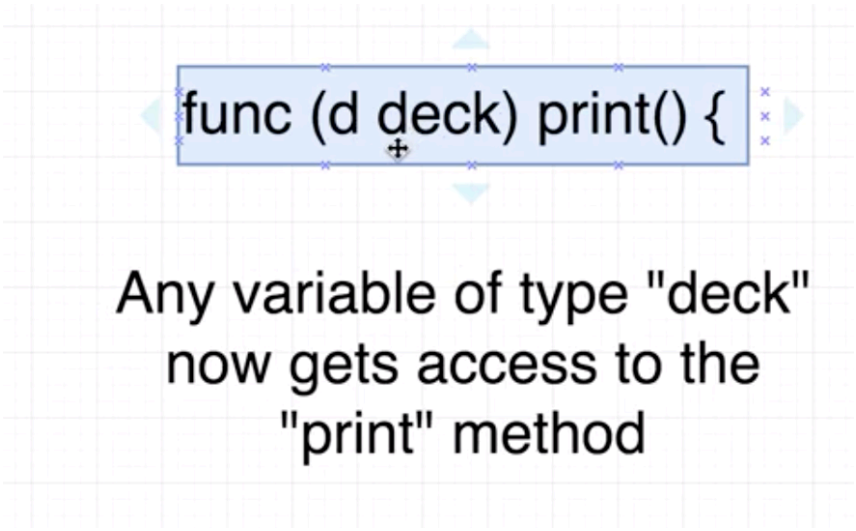
```
type deck []string

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

```

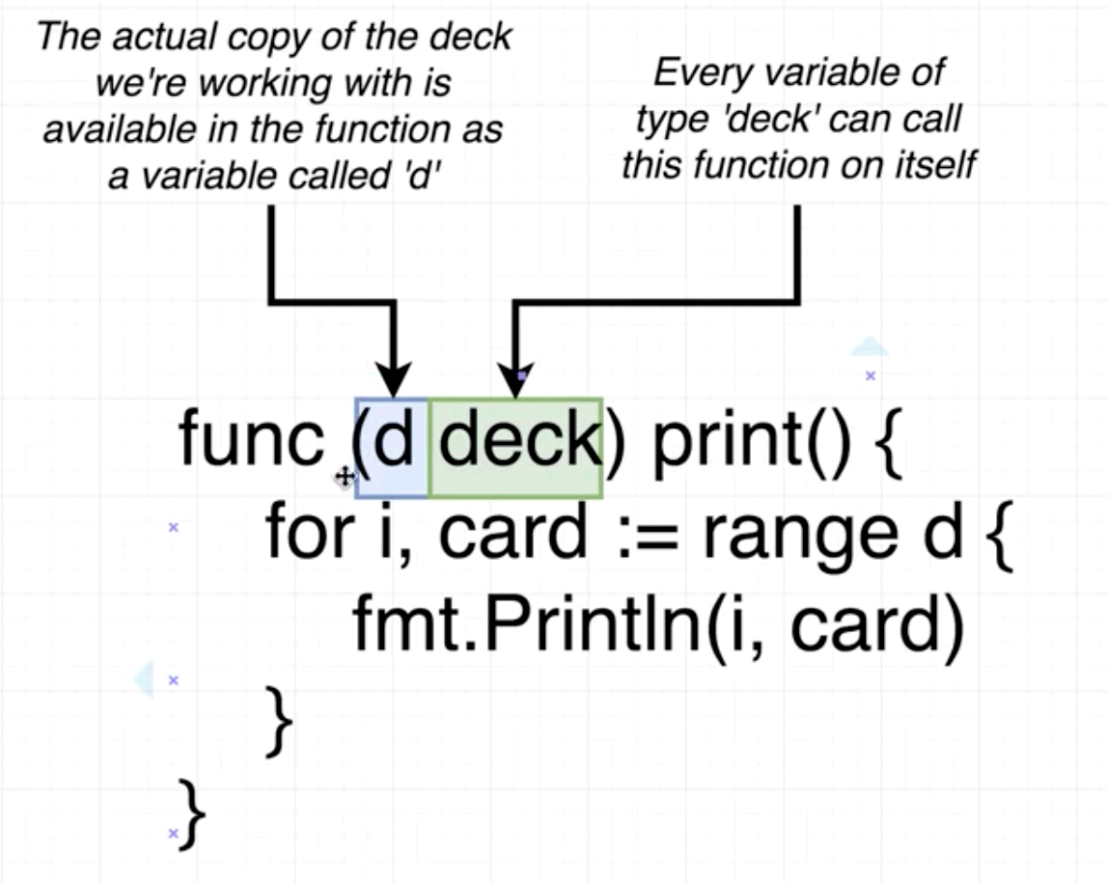
/src/cards/main.go

```
package main

func main() {
    cards := deck{"Ace of Diamonds", newCard()}
    cards = append(cards, "Six of Spades")

    cards.print()
}

func newCard() string {
    return "Five of Diamonds"
}
```

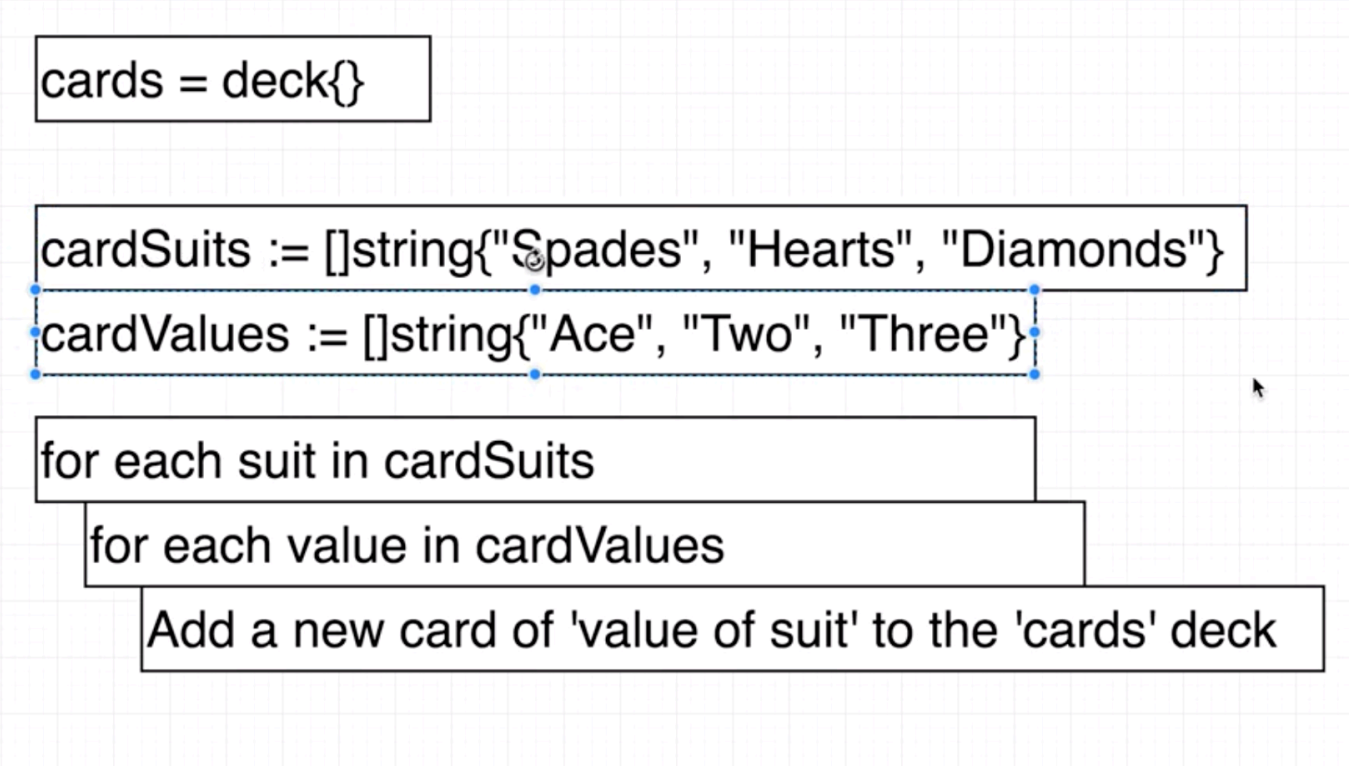
```
$ go run main.go deck.go
```

Receiver Functions





Creating a New Deck



/src/cards/deck.go

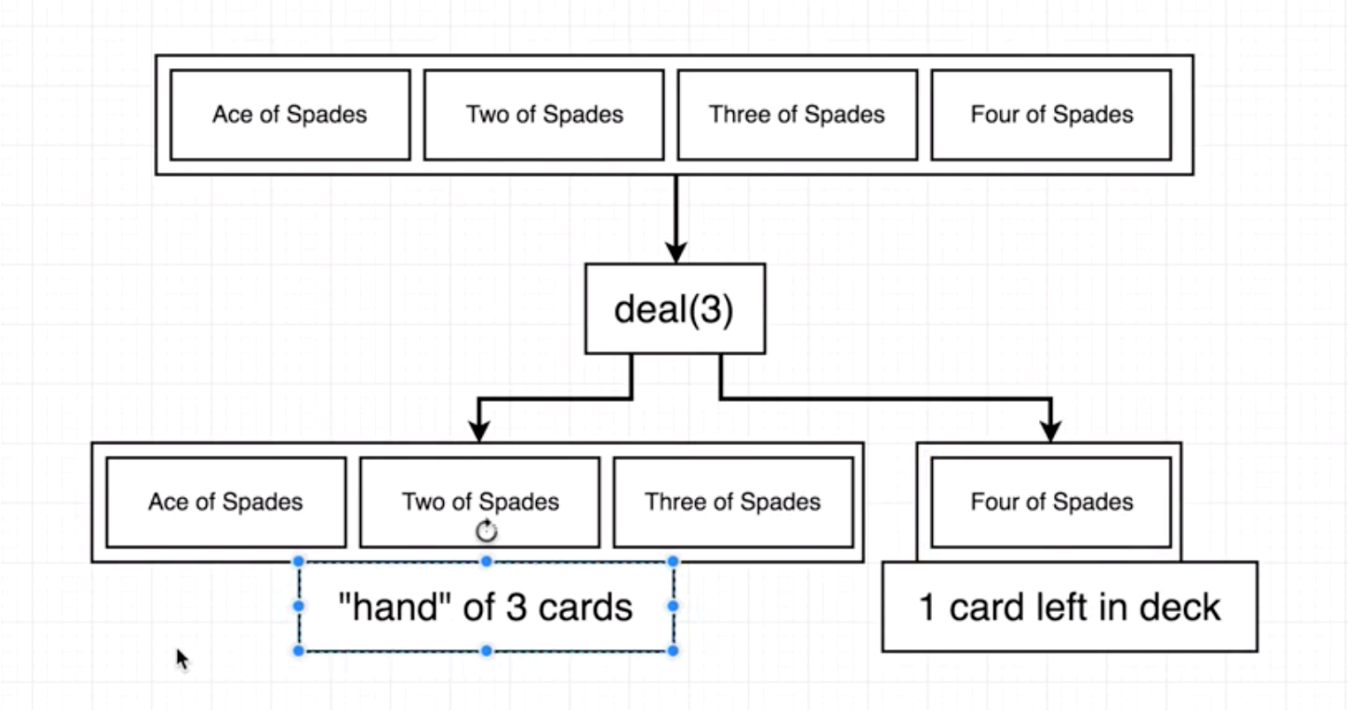
```
1 // Create a new type of 'deck'
2 // which is a slice of strings
3 type deck []string
4
```

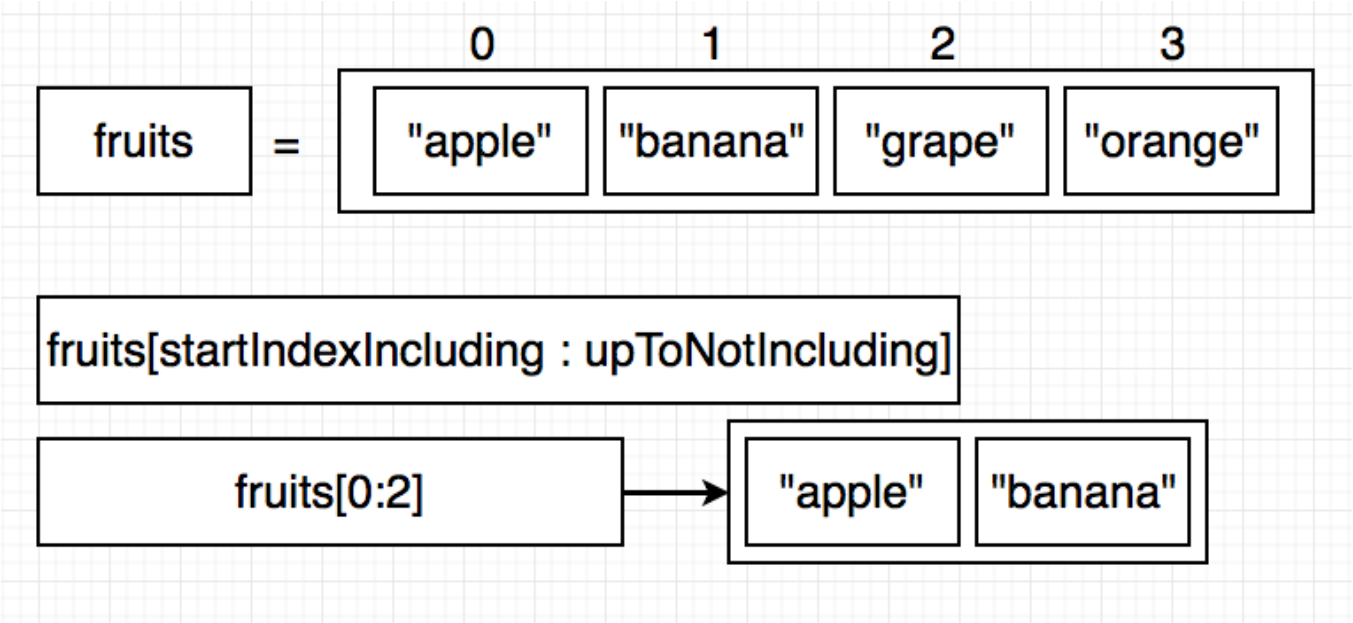
```
5  func newDeck() deck {
6      cards := deck{}
7
8      cardSuits := []string{"Spades", "Diamonds", "Hearts", "Clubs"}
9      cardValues := []string{"Ace", "Two", "Three", "Four"}
10
11     for _, suit := range cardSuits {
12         for _, value := range cardValues {
13             cards = append(cards, 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





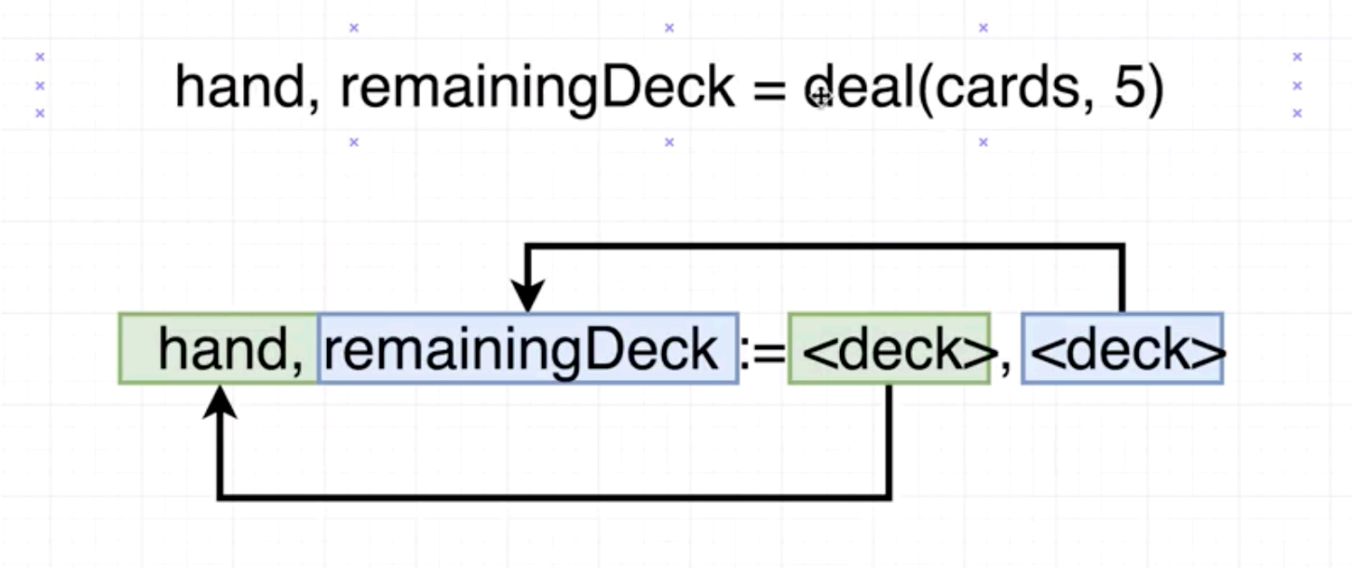
Multipole Return Values

/src/cards/deck.go

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

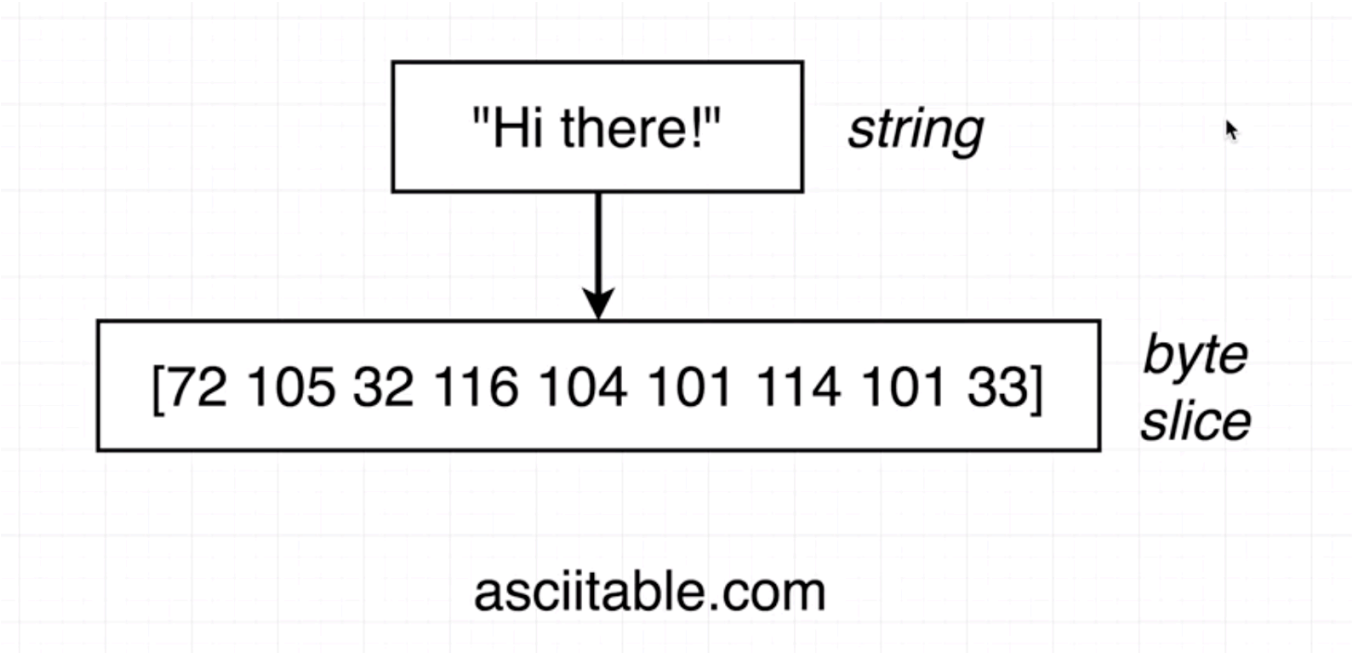
/src/cards/main.go

```
func main() {
    cards := newDeck()
    hand, remainingDeck := deal(cards, 5)

    hand.print()
    remainingDeck.print()
}
```



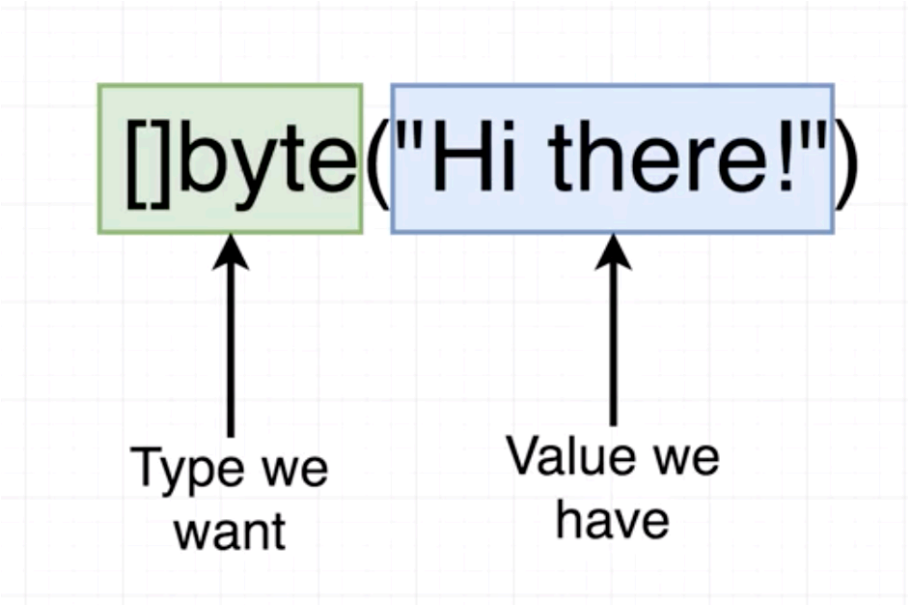
Byte Slices

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



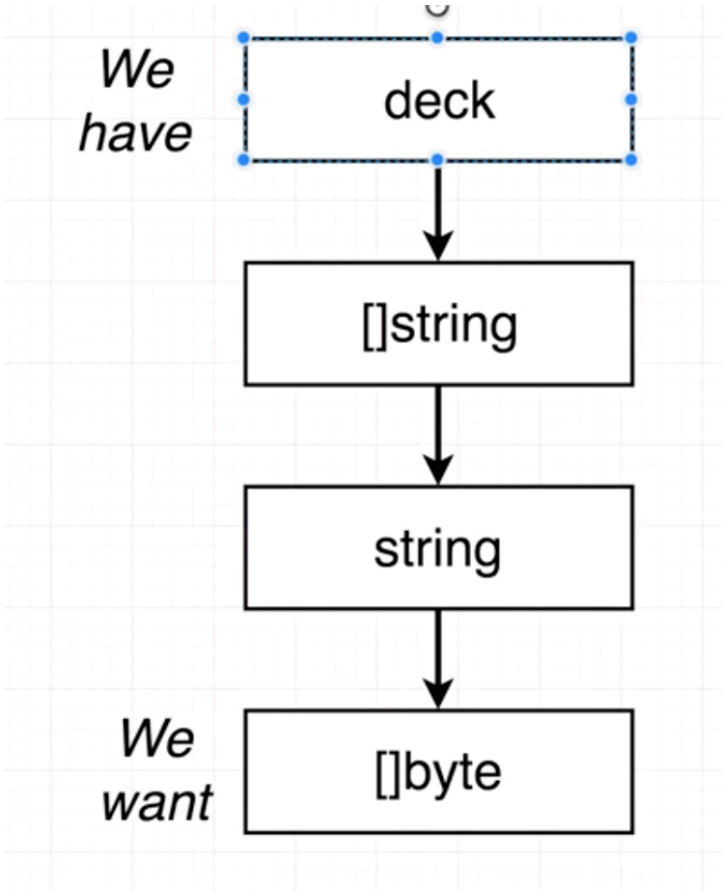
Deck to String

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

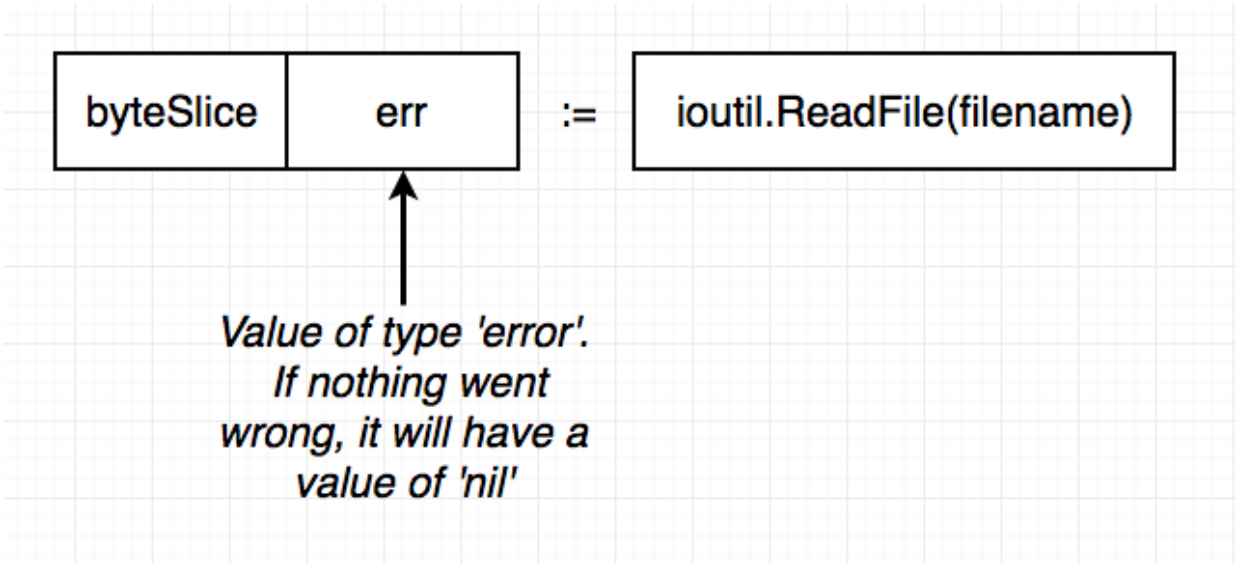
/src/cards/deck.go

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

/src/cards/main.go

```
func main() {
    cards := newDeck()
    cards.saveToFile("my_cards")
}
```

Reading From the Hard Drive



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

/src/cards/deck.go

```
func newDeckFromFile(filename string) deck {
    byteSlice, err := ioutil.ReadFile(filename)
    if err != nil {
        // Option #1 - log the error and return a call to newDeck()
        // Option #2 -log the error and entirely quit the program
        fmt.Println("Error: ", err)
        os.Exit(1)
    }

    stringSlice := strings.Split(string(byteSlice), ",") // Ace of Spades,Two of Spades,Three of Spades,...
    return deck(stringSlice)
}
```

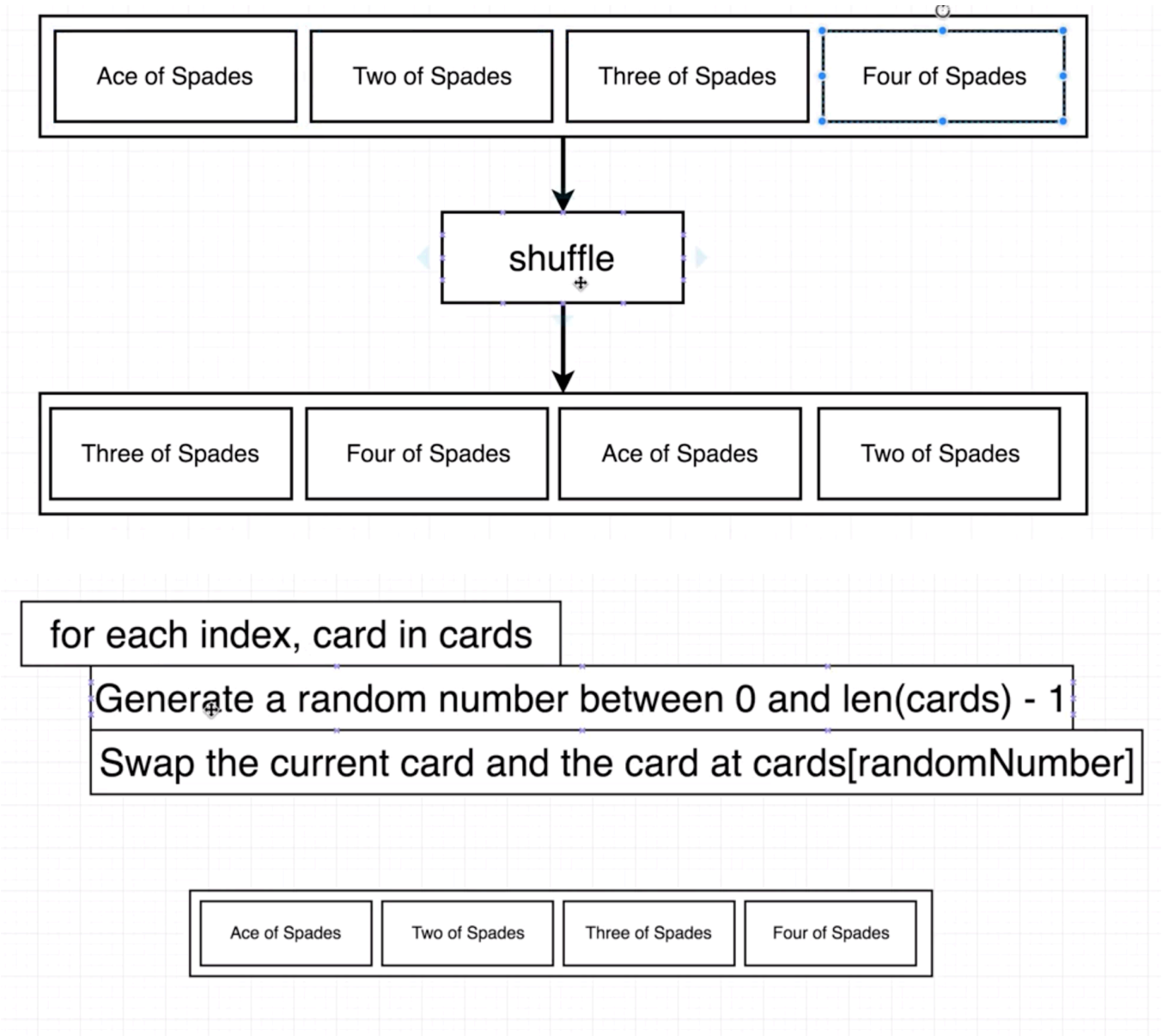
/src/cards/main.go

```
func main() {
    cards := newDeckFromFile("my_cards")
    cards.print()
}
```

/src/cards/main.go (error)

```
func main() {
    cards := newDeckFromFile("my")
    cards.print()
}
```

Shuffling a Deck



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

/src/cards/deck.go

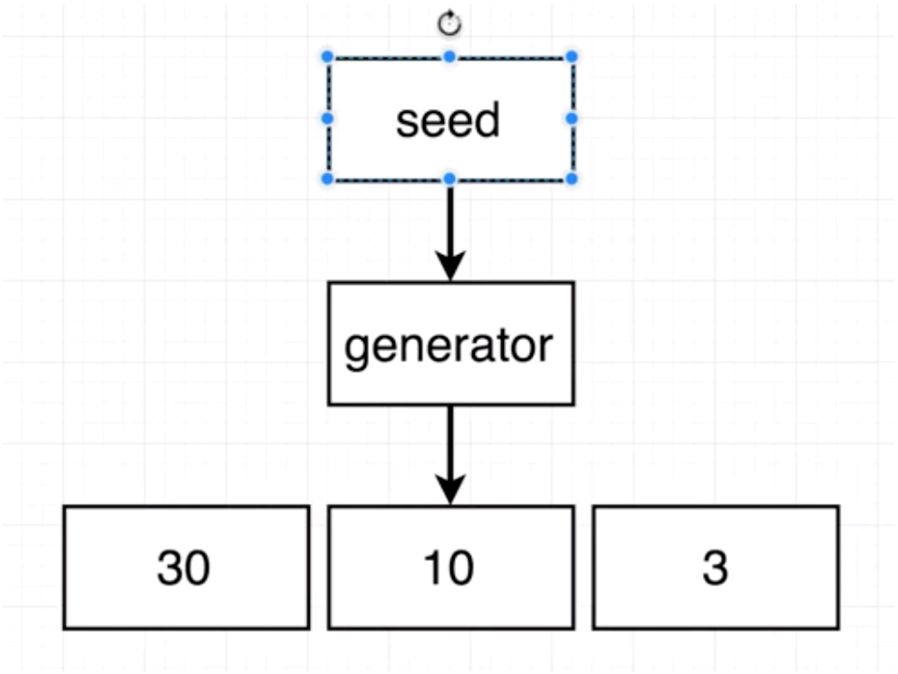
```
func (d deck) shuffle() {
    for index := range d {
        newPosition := rand.Intn(len(d) - 1)
        // swap item in index and in newPosition
        d[index], d[newPosition] = d[newPosition], d[index]

    }
}
```

/src/cards/main.go

```
func main() {
    cards := newDeck()
    cards.shuffle()
    cards.print()
```

```
}
```

Random Number Generation



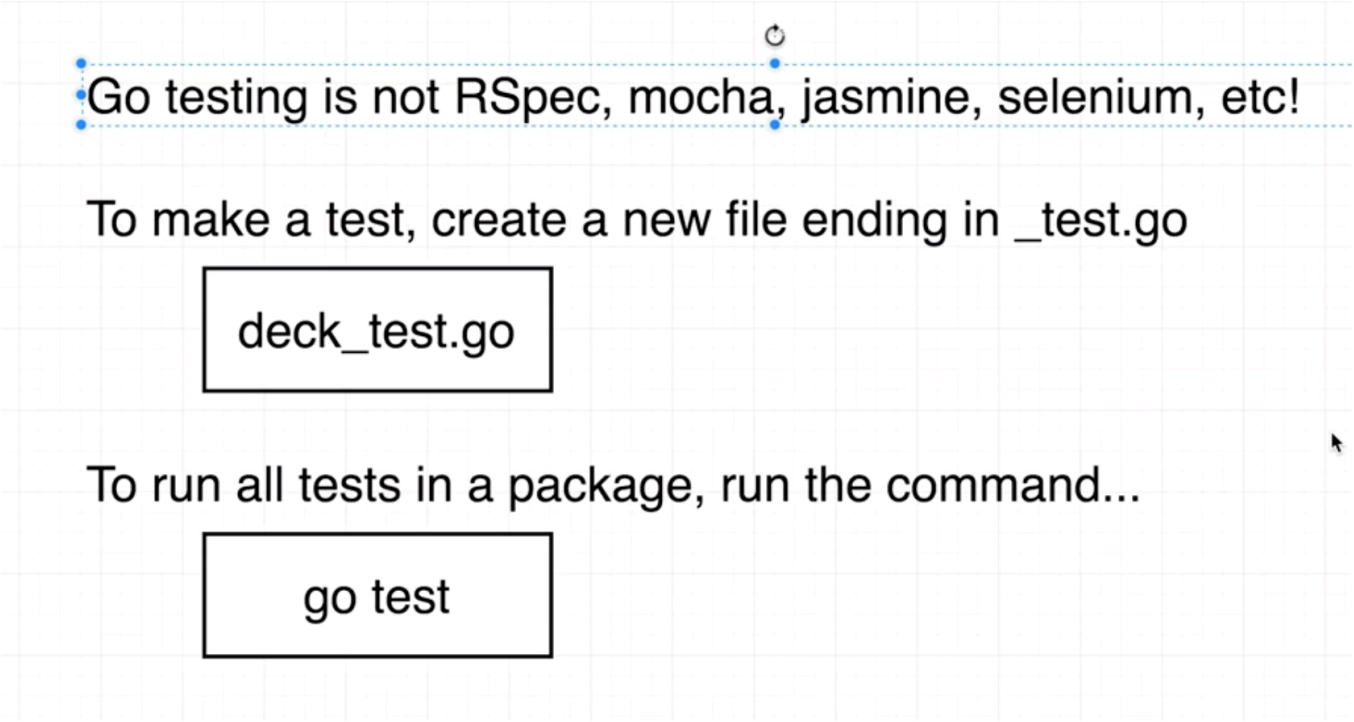
https://golang.org/pkg/math/rand/#Rand

Better Approach to generate random number to change numbers every time

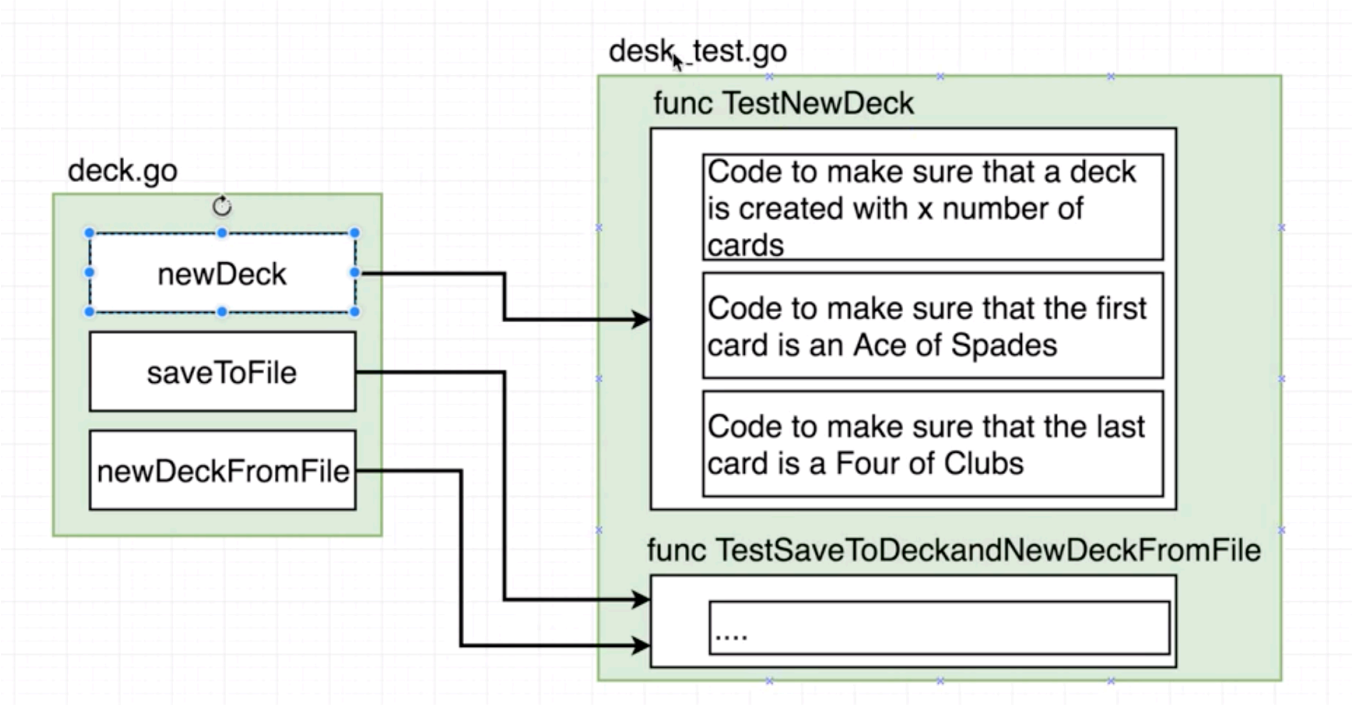
/src/cards/deck.go

```go
func (d deck) shuffle() {
    source := rand.NewSource(time.Now().UnixNano())
    r := rand.New(source)

    for i := range d {
        newPosition := r.Intn(len(d) - 1)
        d[i], d[newPosition] = d[newPosition], d[i]
    }
}
```

Testing With Go



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" {
```

```
        t.Errorf("Expected first card of Ace of Spades, but got %v", d[0])
    }

    if d[len(d)-1] != "Four of Cubes" {
        t.Errorf("Expected last card of Four of Cubes, but got %v", d[len(d)-1])
    }
}
```

Testing File IO

```
func TestSaveToDeckAndNewDeckFromFile(t *testing.T) {
    os.Remove("_decktesting")

    deck := newDeck()
    deck.saveToFile("_decktesting")

    loadedDeck := newDeckFromFile("_decktesting")

    if len(loadedDeck) != 16 {
        t.Errorf("Expected 16 cards in deck, got %v", len(loadedDeck))
    }

    os.Remove("_decktesting")
}
```

Assignment 1: Even and Odd

```
func main() {
    numbers := []int{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

    for number := range numbers {
        if number%2 == 0 {
            fmt.Println(number, " is even")
        } else {
            fmt.Println(number, " is odd")
        }
    }
}
```