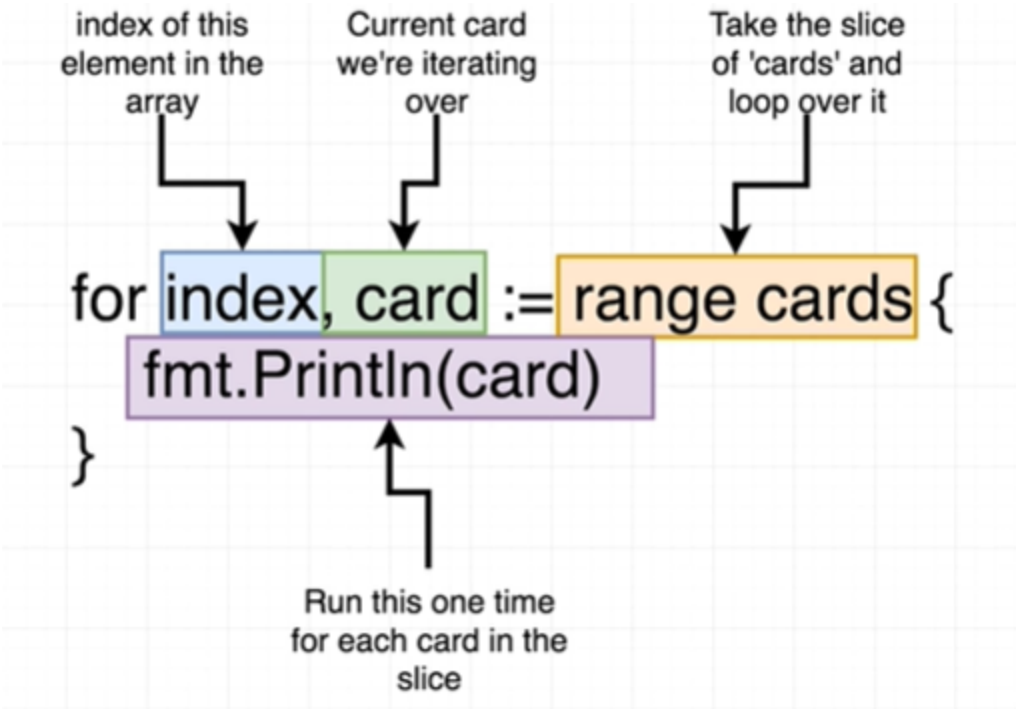
Deeper Into Go

Created	@Feb 1, 2021 5:07 PM
Tags	

Slices

> 💡 **Slice:** an array that can grow or shrink (a la *ArrayList*)

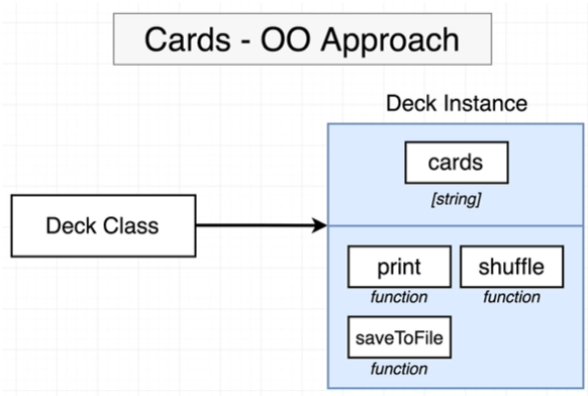
- Slice Declaration:
  - `cards := []string{"Ace of Diamonds", newCard()}`
- How to iterate over a slice:



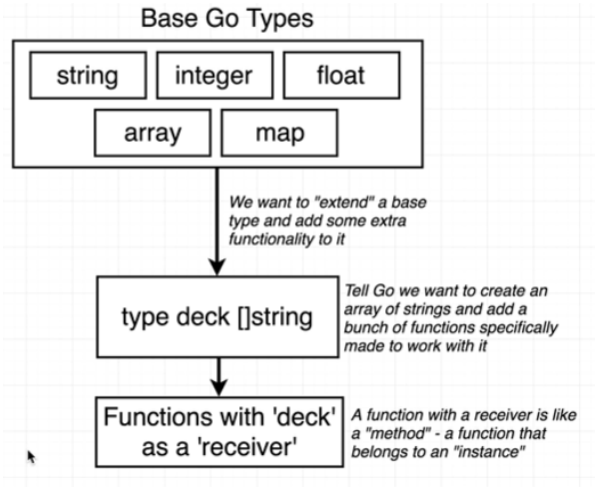
OO Approach vs Go Approach

- Go is **not** an object-oriented languages

OO Approach



Go Approach



- In `deck.go` the line `type deck []string` creates a new type *deck* that is a slice of strings

---

Receiver Functions

- The leftward code defines a **receiver function** of the **type** *deck*
- By convention, the reference to the receiver is a one or two letter abbreviation that refers to the type
  - In this case `d` refers to a **value** of type *deck*- which is comparable to the term `this` in OO languages

```
type deck []string

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

The choice to use a receiver function (`cards.myFunction()`) as opposed to a function with a value passed in as a parameter (`myFunction(cards)`) is somewhat stylistic

Convention says that receiver functions should generally affect the value they're called against as opposed to simple using it to find some other data

---

Creating a New Deck

- This `newDeck` function does not need to be a receiver because we are not using the value of a variable
- When we call it, we need only put `cards := newDeck()` no `.` needed

```
func newDeck() deck {
  cards := deck{}

  cardSuits := []string{"Spades", "Hearts", "Clubs", "Diamonds"}
  cardValues := []string{"Ace", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight",
"Nine", "Ten", "Jack", "Queen", "King"}

  for _, suit := range cardSuits {
    for _, value := range cardValues {
      cards = append(cards, value+" of "+suit)
    }
  }

  return cards
}
```


Use an underscore `_` to replace a variable that is declared but you don't want to use:

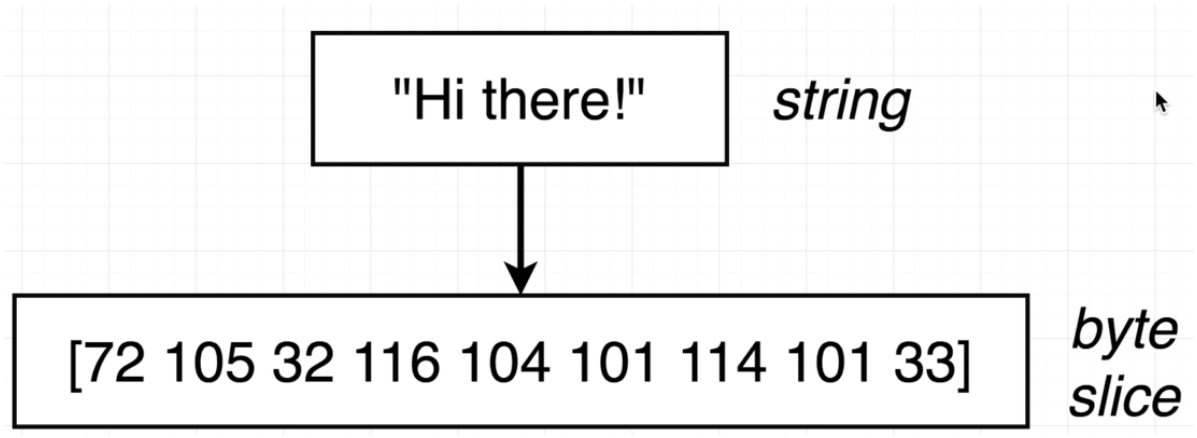
```
for _, value := range cardValues {
    cards = append(cards, value+" of "+suit)
  }
```

Slice Range Syntax

- Go has a built in syntax to take a range out of a slice/array
  `fruits[ startIndexIncluding : upToNotIncluding ]`
- There is a short hand for going from the start, or going to the end
  - `fruits[:3]` will yield the first 3 elements at indices 0, 1, and 2
  - `fruits[3:]` will yield everything but the first 3 elements (at indices 3, 4, 5, ...)

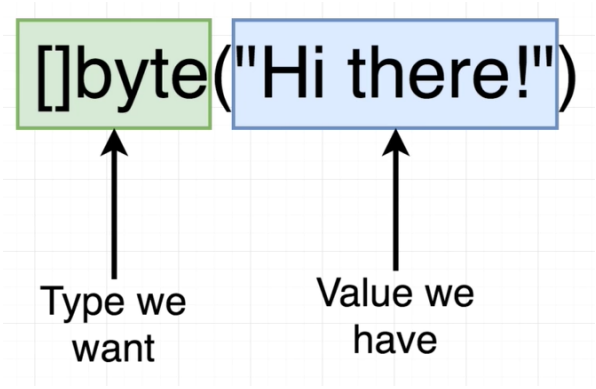
Byte Slices

- `io/ioutil` package in Go contains filesystem access
  - Contains `WriteFile` and `ReadFile` functions which expect a **byte slice**



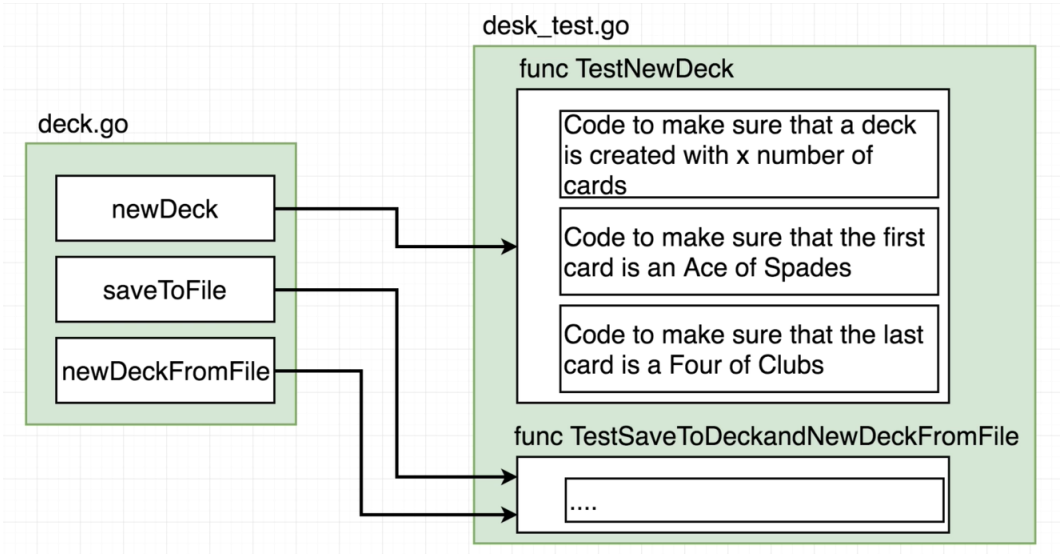
Type Conversion

- Types can be changed with this format:



Testing With Go

- filenames ending with `_test.go` allow you to run portions of go code



- The convention is to name test functions in ***PascalCase*** and normal functions in ***camelCase***