**Slice:-**

how to write slice:-

g=> cards := []string {"ace of diamond", "ace of spade"}

so this is slice of of **string** (not string**s**) and in curly braces we define its element.

we can do g=> fmt.Println(cards)

Also how can we append slice (add new item)

g=> cards = append(cards, "sixth of heart")

this append function will return a new slice and that new slice will overwrite card variable.

so this can we write as

g=> cards2 := append(cards1, "sixth of heart")

so there cards1 and cards2 are two different slice.

**append function will not modify argumented slice**

func main() {

    cards := []string{"five of diamond", newCard()}

    cards2 := append(cards, "six of heart")

    fmt.Println(cards)

    fmt.Println(cards2)

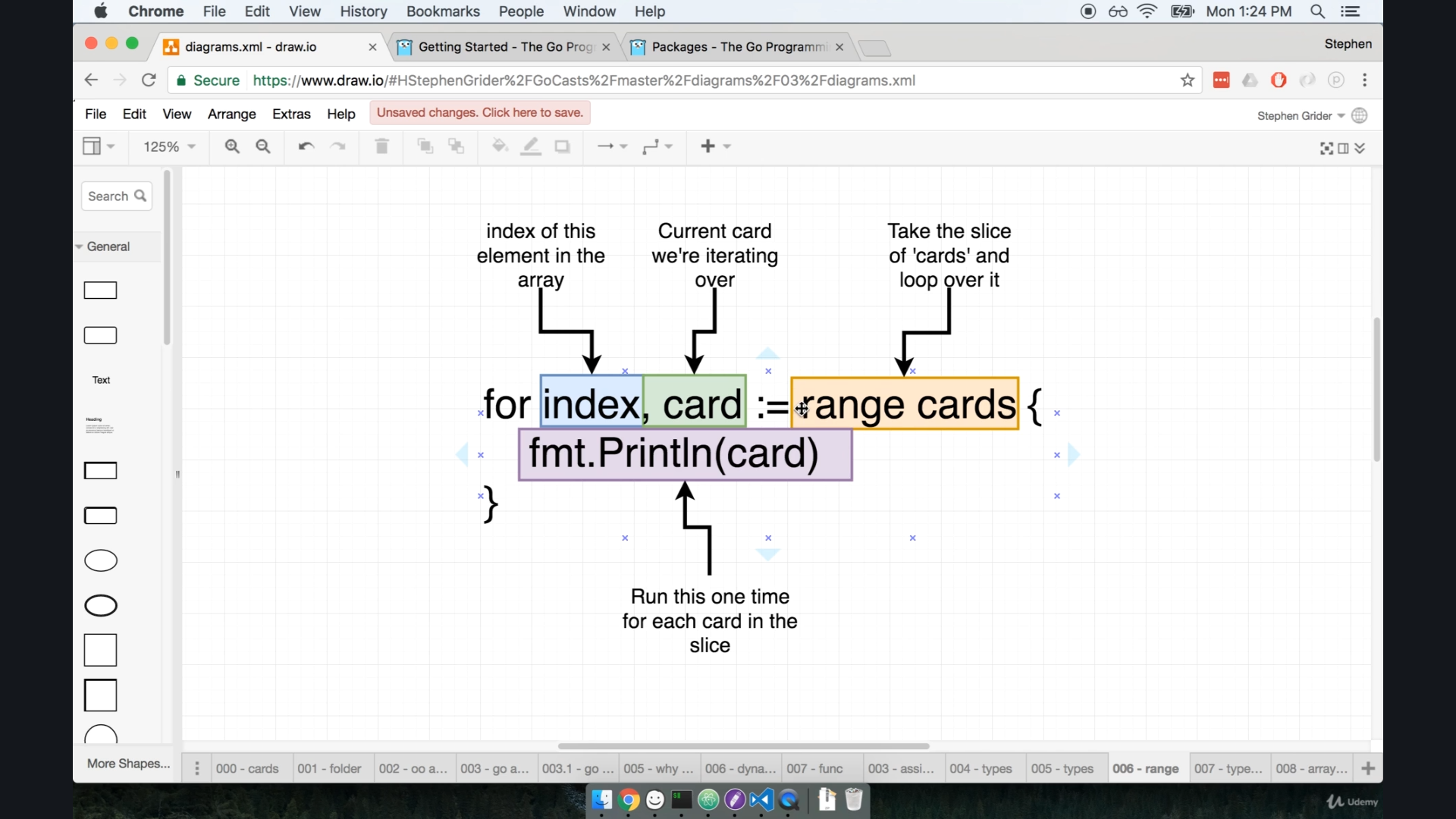
}

func newCard() string {

    return "fifth of diamond"

}

So how can we iterate over a slice:-



**range** => range is a **keyword** that are used when we want to iterate over every single element in slice.

we use ":=" as index and card but is that already declare for next iteration ?

No, for each iteration index and card are thrown or destroy and initialize freshly.

for **for** loop we use ":="

func main() {

    cards := []string{"five of diamond", newCard()}

    cards = append(cards, "six of heart")

    for i, card := range cards {

        fmt.Println(i, card)

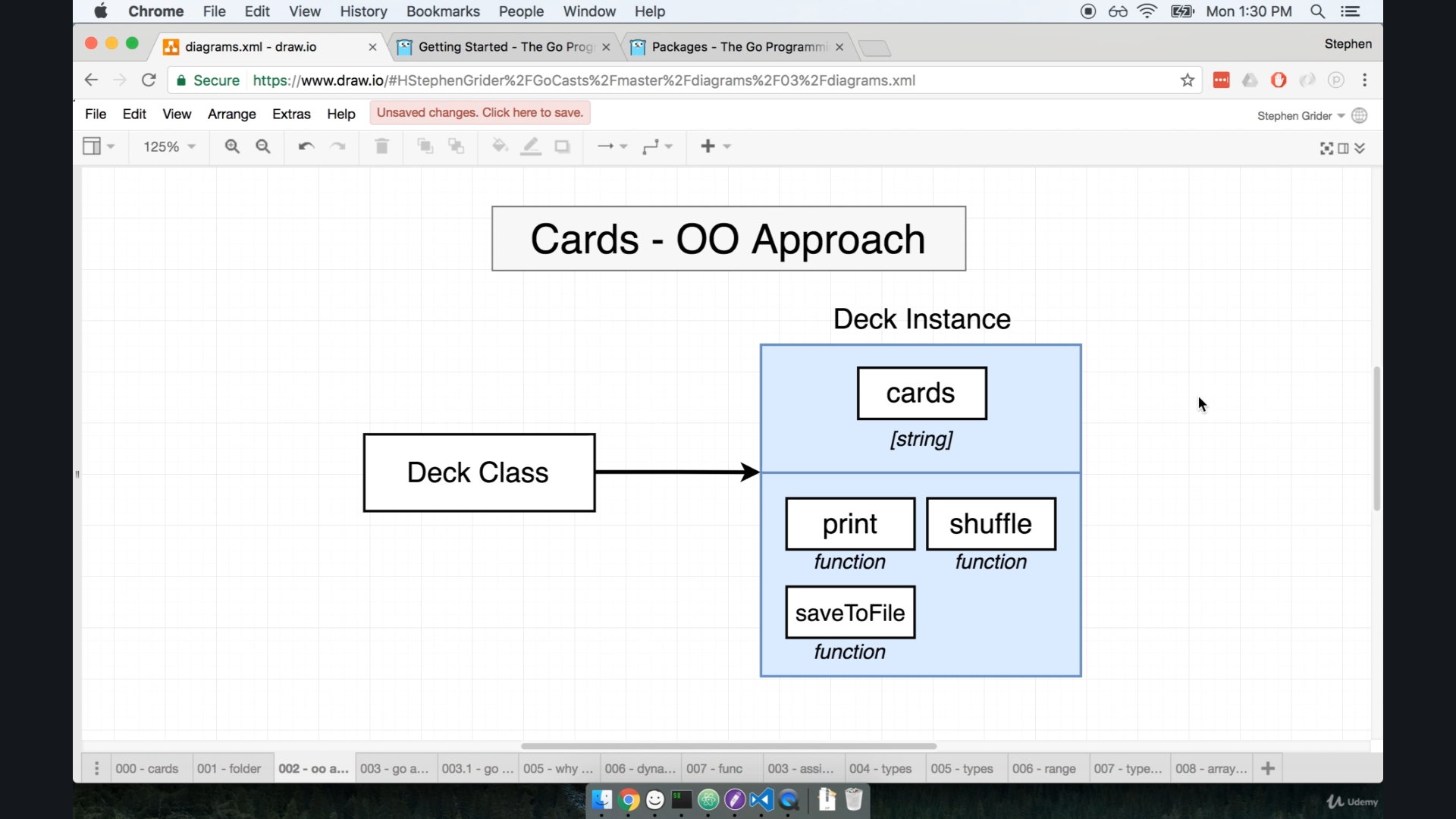
    }

}

Go is **not** an OOP language

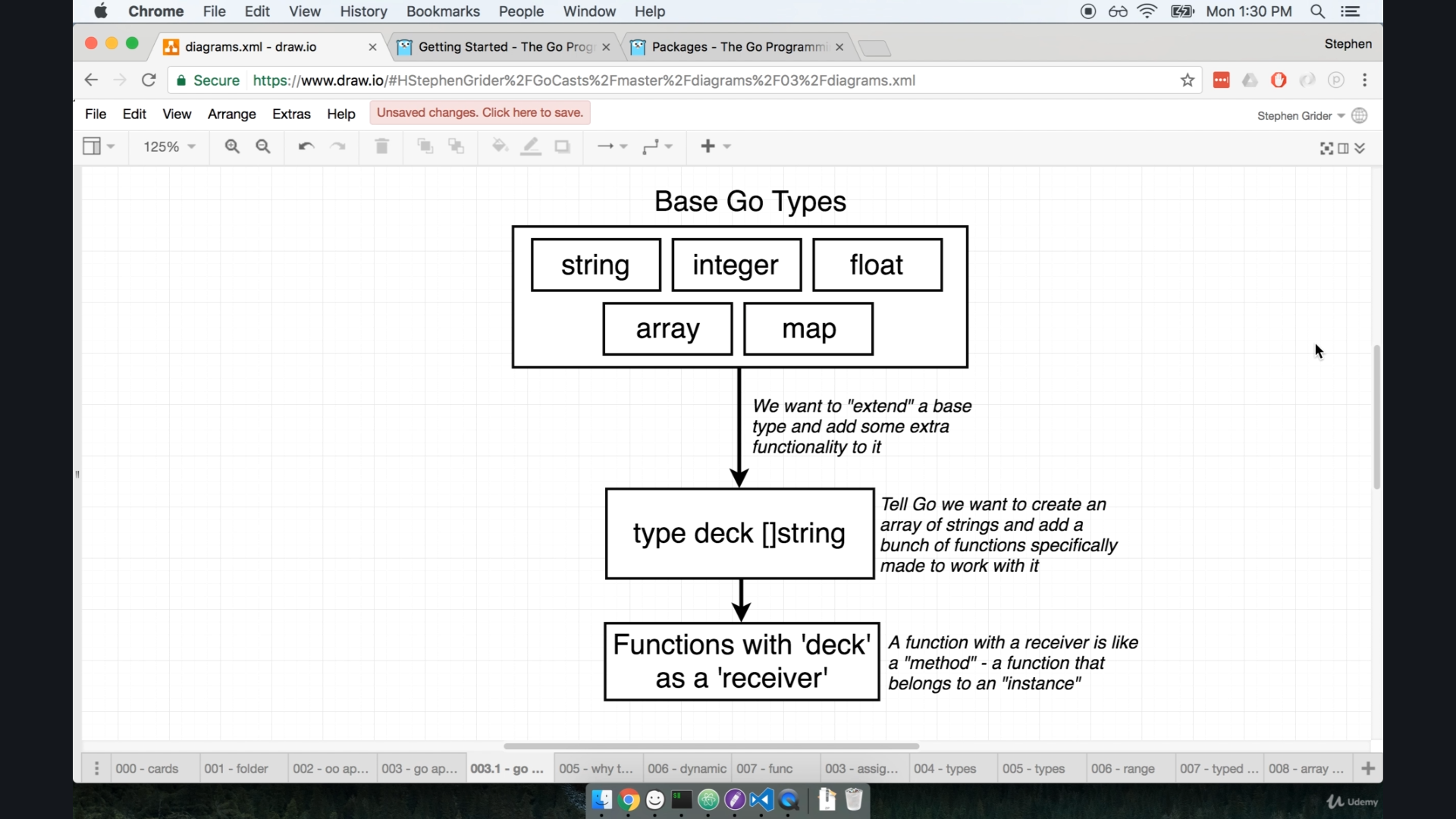
So there is no idea of classes inside Go.

lets look how our deck card project in both oops and golang context



here we can see oops approach like we have first deck class and as we know class contain variables and method init.

Now come on golang



for golang we know that we have some basic types are available to us.

now are going to define a **new custom datatype** called as **"deck"** which basically contain slice of string (name of cards)

now we are going to **attach some function** to this **new custom datatype** (deck) and these function is refer as **function as a receiver**

we will discuss this function as a receiver in detail later.

but the a main thing is that **indirectly,** we are extending functionality of preemptive datatype (like in this case string).

Now we are going to create **custom datatype** :-

now create "deck.go" file inside our project, and its first line will be "package main"

in this file we are going to declare our custom datatype.

**g=> type deck []string => so deck is just slice of string but when we define like that deck can bind some function into it.**

**main.go**

package main

import "fmt"

func main() {

    cards := deck{"6 spades", "2 heart", newCard()}

    for index, card := range cards {

        fmt.Println(index, card)

    }

}

func newCard() string {

    return "fifth of diamond"

}

**deck.go**

package main

type deck []string

**>** go run main.go deck.go

here we can see that package name is "main".

also till now we have seen "deck" is working as alternate to "[]string".

now we will **write a function that belong to deck** datatype and this will print all cards.

syntax:- func (receiver) function-name (args) {

body

}

example:-

func **(d deck)** print(){

for I, card := range d { // this is **d -> “object of deck”, received when calling this fund.**

fmt.Println(I, card)

}

}

where **(d deck)** will be a receiver.

**main.go**

package main

func main() {

    cards := deck{"6 spades", "2 heart", newCard()}

    cards.print()

}

func newCard() string {

    return "fifth of diamond"

}

**deck.go**

package main

import "fmt"

type deck []string

func (d deck) print() {

    for i, card := range d {

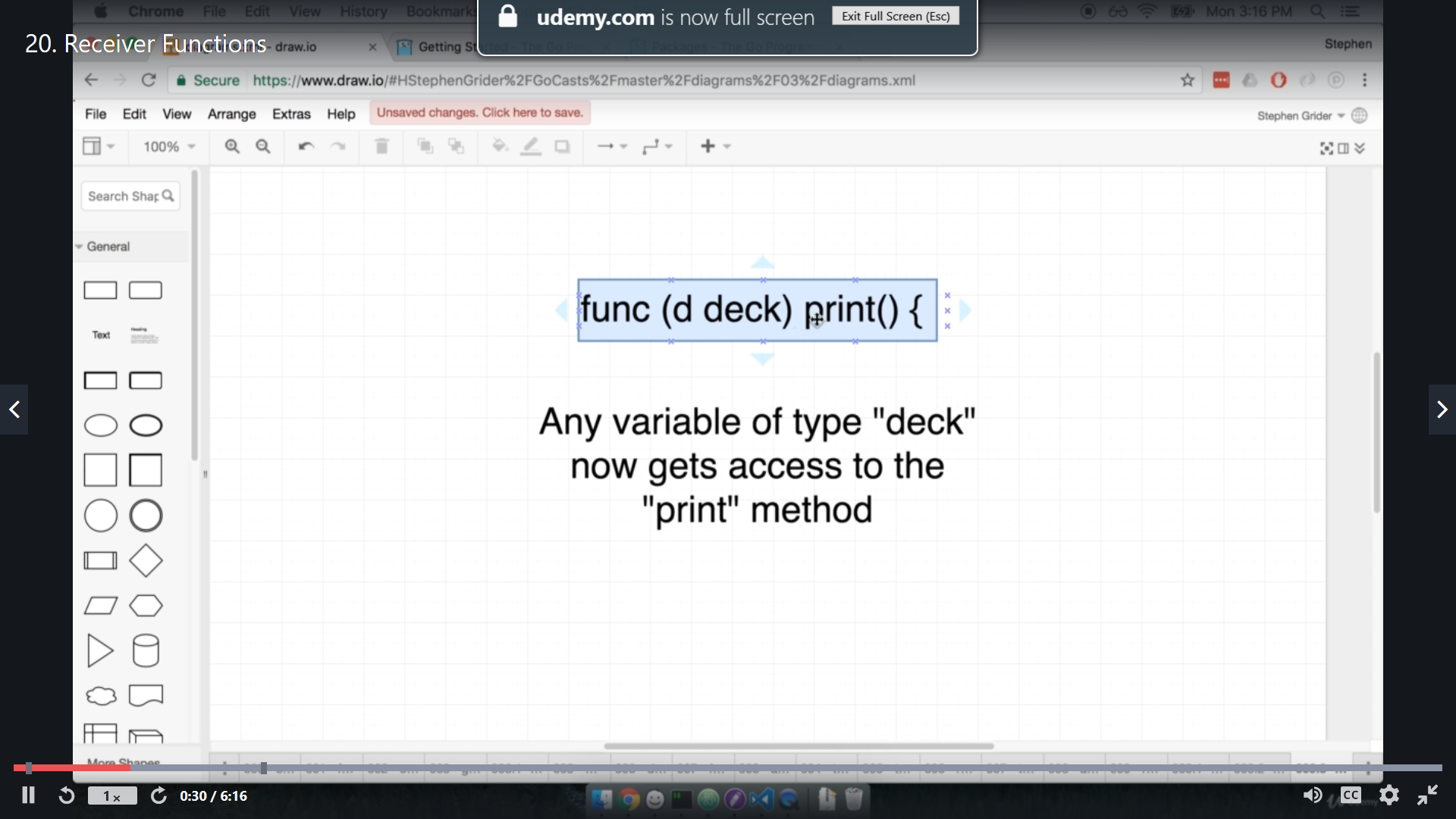
        fmt.Println(i, card)

    }

}

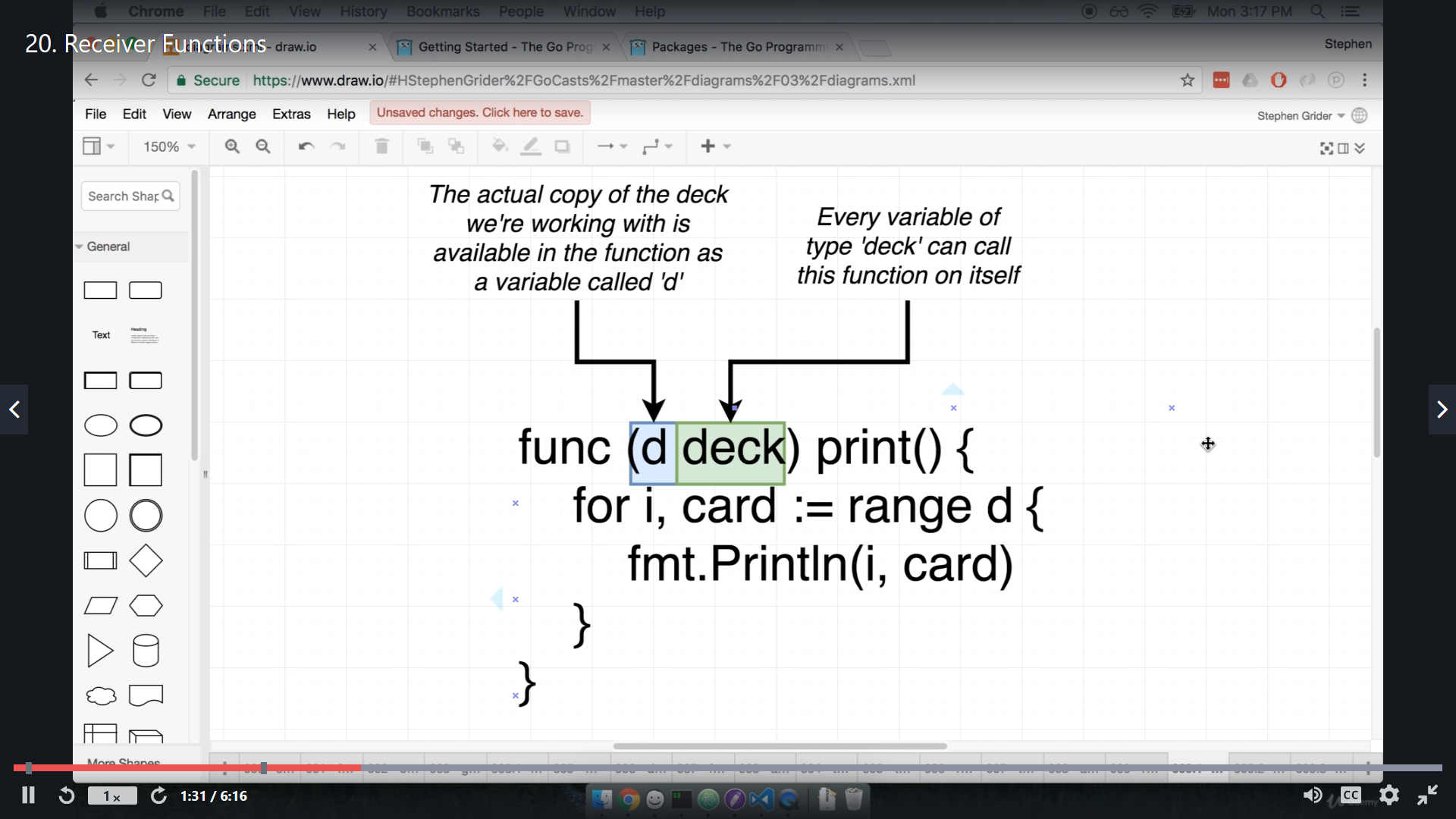
> go run main.go deck.go

now we will discuss syntax we here.



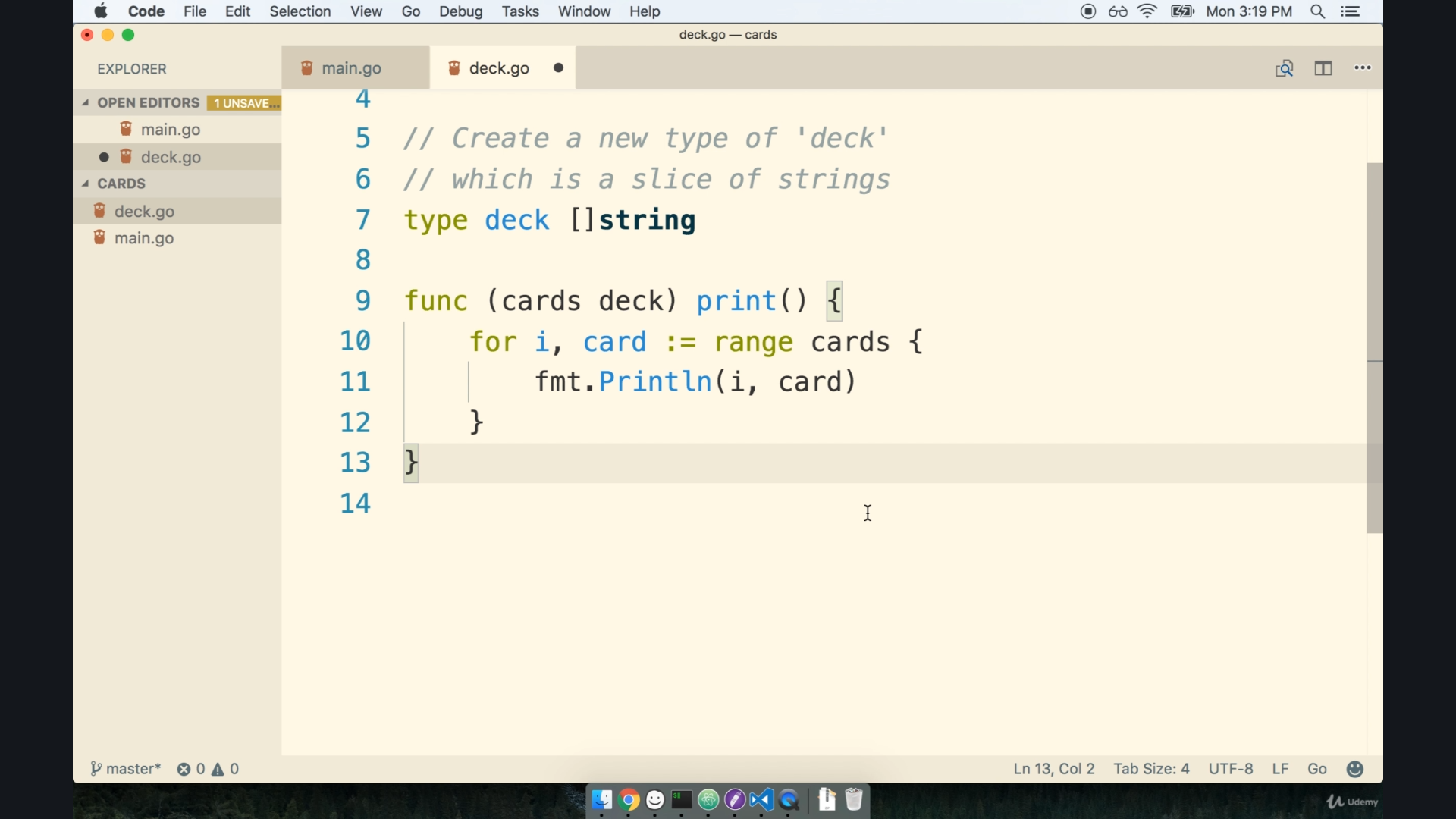
this **(d deck)** is knows as **receiver** to this print function

in cards.print() , “cards” can be access as “d”, where “cards” is object of “deck”, which is pass as receiver, so that d take cards’s value for this case.



"d" refer to "instance" of "deck" variable in our case it is "cards".

also an **analogy** to this "**d**" with other language is like **self** , **this**



we can also put the reference name other than "d" like above it is "cards".

but by convention we put “d” because it is first letter of custom datatype name "deck".

and give more general infatuation for many variable of deck.

so this is **similar** to class in object oriented approach.

now lets create deck of cards ie putting all card name into string slice.

but doing that manually is tedious, we are going to use nested for loop to concatinate string and put it into string slice.

create a function "newDeck() deck " in deck.go file.

func newDeck() deck {

    var newCards deck

    cardVal1 := []string{"spade", "heart", "diamond"}

    cardVal2 := []string{"ace", "two", "three", "four"}

    for \_, val1 := range cardVal1 {

        for \_, val2 := range cardVal2 {

            newCards = append(newCards, val1+" of "+val2)

        }

    }

    return newCards

}

func (d deck) print() {

    for i, card := range d {

        fmt.Println(i, card)

    }

}

for \_, val1 => here "\_" in place of index because this means we know that here something to be return but we don’t want to use in our code.

func main() {

newDeck := newDeck()

newDeck.print()

}

func main() {

newDeck := newDeck()

newDeck.print()

}