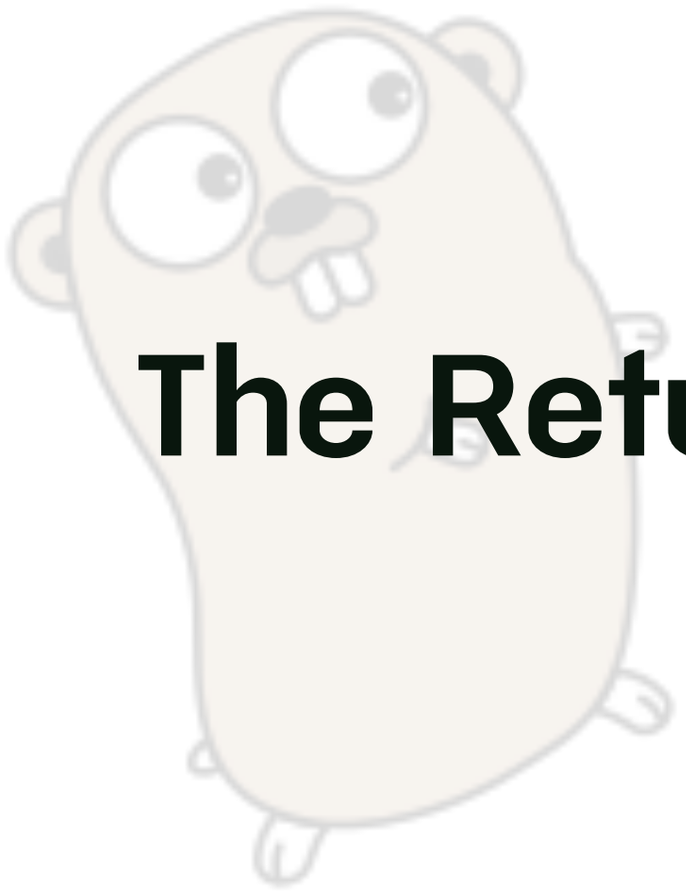


The Return of Vending Machine

Vending-machine

Vending Machine
Coin: TEN(10), Five(5), TWO(2), ONE(1)
T F TW O

Item: Soft Drink(18),
Canned Coffee(12),
Drinking Water(7)

Coin Return: returns all inserted money

#Criteria
Unlimited items
Unlimited change
Currently inserted money

Vending-machine

1. Buy SD(soft drink) with exact change
Insert: T, F, TW, O
Currently inserted money: 18
Choose: Select SD
Return: SD

2. Start adding change but hit coin return
Insert: T, T, F
Currently inserted money: 25
Choose: Coin Return
Return: T, T, F

3. Buy CC(canned coffee) without exact change
Insert: T, T
Currently inserted money: 20
Choose: Select CC
Return: CC, F, TW, O

Vending-machine

List:

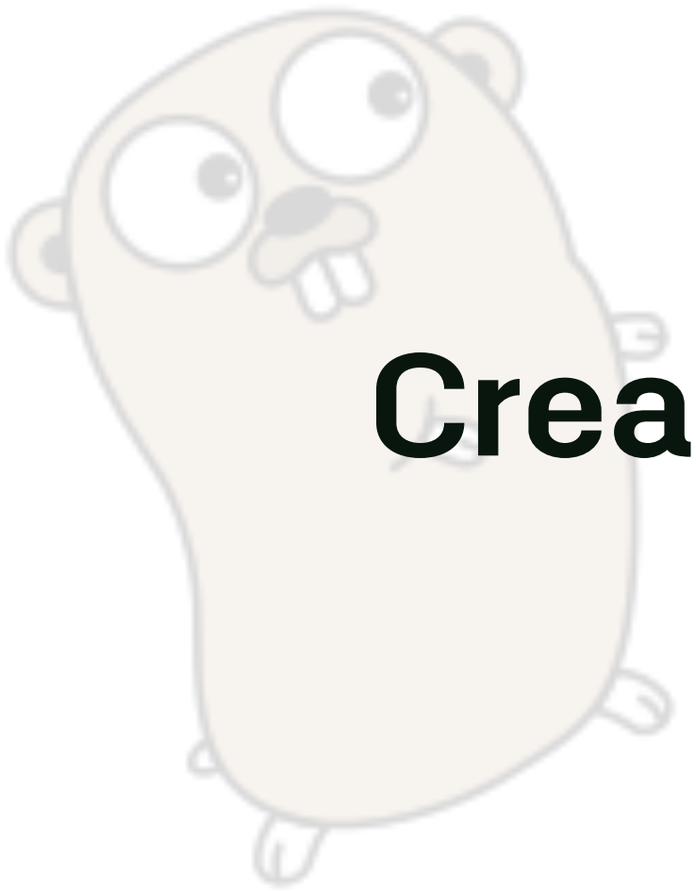
[ ] when **vending machine** is created the **money** should be “**zero**”

[ ] when we **adding coin** then we should know how much **money** we have added

[ ] after **adding coins** with **exact change** when **select item** the output should show Item without change

[ ] if we **adding coins without exact change** when **select item** the **output should include change**

[ ] if we **adding coins** and choose “**Return Coin**” then the change should **equals** the value that we have added



Create vending machine

Vending-machine

[x] when vending machine is created the inserted money should be “zero”


```go
type VendingMachine struct {
}

func (m VendingMachine) InsertedMoney() int {
    return 0
}

func main() {
    vm := VendingMachine{}
    fmt.Println("Inserted Money:", vm.InsertedMoney())
    // Inserted Money:  0
}
```

Add coin

Vending-machine

[ ] when we adding coin then we should know how much money we have added


```go
func (m VendingMachine) InsertedMoney() int {
    return 10
}

func (m *VendingMachine) InsertCoin(coin string) {
}

func main() {
    vm := VendingMachine{}
    fmt.Println("Inserted Money:", vm.InsertedMoney())
    // Inserted Money:  0
    vm.InsertCoin(“T")
    fmt.Println("Inserted Money:", vm.InsertedMoney())
    // Inserted Money: 10
}
```

We do hard code to make it work.

Is it right?

Vending-machine

[ ] when we adding coin then we should know how much money we have added


```go
type VendingMachine struct {
    insertedMoney int
}

func (m VendingMachine) InsertedMoney() int {
    return m.insertedMoney
}

func (m *VendingMachine) InsertCoin(coin string) {
    m.insertedMoney = 10
}

func main() {
    vm := VendingMachine{}
    fmt.Println("Inserted Money:", vm.InsertedMoney())
    // Inserted Money:  0
    vm.InsertCoin(“T")
    fmt.Println("Inserted Money:", vm.InsertedMoney())
    // Inserted Money: 10
}
```

Add other coins

Vending-machine

[ ] when we adding coin then we should know how much money we have added


```go
func (m *VendingMachine) InsertCoin(coin string) {
    if coin == “T" {
        m.insertedMoney += 10
    }
    if coin == “F" {
        m.insertedMoney += 5
    }
}

func main() {
    vm := VendingMachine{}
    fmt.Println("Inserted Money:", vm.InsertedMoney())
    // Inserted Money:  0
    vm.InsertCoin("T")
    vm.InsertCoin("F")
    fmt.Println("Inserted Money:", vm.InsertedMoney())
    // Inserted Money: 15
}
```

Vending-machine

[ ] when we adding coin then we should know how much money we have added


```
func (m *VendingMachine) InsertCoin(coin string) {
    if coin == "T" {
        m.insertedMoney += 10
    }
    if coin == "F" {
        m.insertedMoney += 5
    }
    if coin == “TW" {
        m.insertedMoney += 2
    }
}

func main() {
    vm := VendingMachine{}
    fmt.Println("Inserted Money:", vm.InsertedMoney())
    // Inserted Money:  0
    vm.InsertCoin("T")
    vm.InsertCoin("F")
    vm.InsertCoin("TW")
    fmt.Println("Inserted Money:", vm.InsertedMoney())
    // Inserted Money: 17
}
```

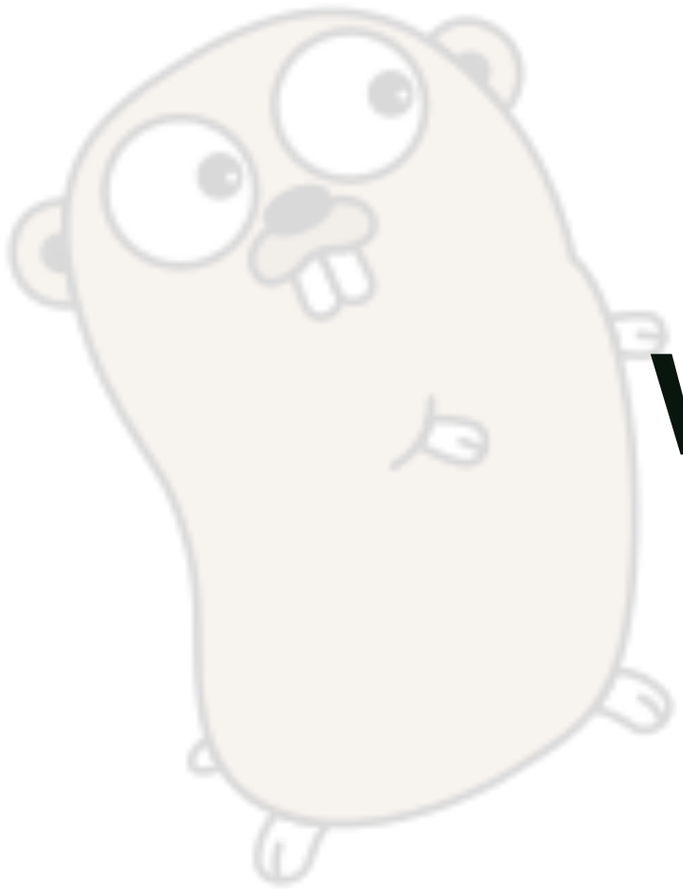
Have you seen the pattern?

[Key : Value]

Vending-machine

[ ] when we adding coin then we should know how much money we have added


```go
type VendingMachine struct {
    insertedMoney int
    coins         map[string]int
}
func (m *VendingMachine) InsertCoin(coin string) {
    m.insertedMoney += m.coins[coin]
}

func main() {
    vm := VendingMachine{}
    fmt.Println("Inserted Money:", vm.InsertedMoney())
    // Inserted Money:  0
    vm.InsertCoin("T")
    vm.InsertCoin("F")
    vm.InsertCoin("TW")
    fmt.Println("Inserted Money:", vm.InsertedMoney())
    // Inserted Money: 17
}
```



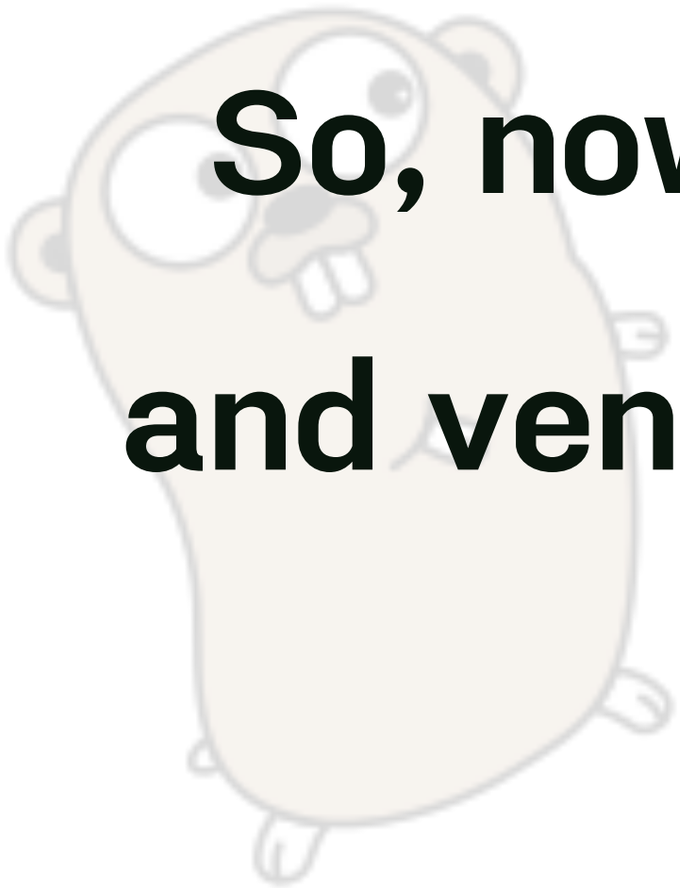
What’s happen?

nil

Vending-machine

[x] when we adding coin then we should know how much money we have added


```go
type VendingMachine struct {
    insertedMoney int
    coins         map[string]int
}
func (m *VendingMachine) InsertCoin(coin string) {
    m.insertedMoney += m.coins[coin]
}

func main() {
    var coins = map[string]int{"T": 10, "F": 5
            , "TW": 2, "O": 1}
    vm := VendingMachine{coins: coins}
    fmt.Println("Inserted Money:", vm.InsertedMoney())
    // Inserted Money:  0
    vm.InsertCoin(“T")
    vm.InsertCoin(“F")
    vm.InsertCoin(“TW")
    fmt.Println("Inserted Money:", vm.InsertedMoney())
    // Inserted Money: 17
}
```



So, now we can add any coin and vending machine can show inserted money

Buy Item with exact change

Vending-machine

[ ] after **adding coins** with **exact change** when **select item** the output should show Item without change


```go
func (m *VendingMachine) SelectSD() string {
    return "SD"
}

func main() {
    var coins = map[string]int{"T": 10, "F": 5
                , "TW": 2, "O": 1}
    vm := VendingMachine{coins: coins}
    vm.InsertCoin(“T")
    vm.InsertCoin(“F")
    vm.InsertCoin(“TW")
    fmt.Println("Inserted Money:", vm.InsertedMoney())
    // Inserted Money: 17
    can := vm.SelectSD()
    fmt.Println(can) // SD
}
```

Vending-machine

[ ] after **adding coins** with **exact change** when **select item** the output should show Item without change


```go
func (m *VendingMachine) SelectSD() string {
    return "SD"
}
func (m *VendingMachine) SelectCC() string {
    return "CC"
}

func main() {
    …
    fmt.Println("Inserted Money:", vm.InsertedMoney())
    // Inserted Money: 17
    can = vm.SelectSD()
    fmt.Println(can) // SD
    vm.InsertCoin("T")
    vm.InsertCoin("TW")
    fmt.Println("Inserted Money:”, vm.InsertedMoney())
    // Inserted Money: 12
    can = vm.SelectCC()
    fmt.Println(can) // CC
}
```

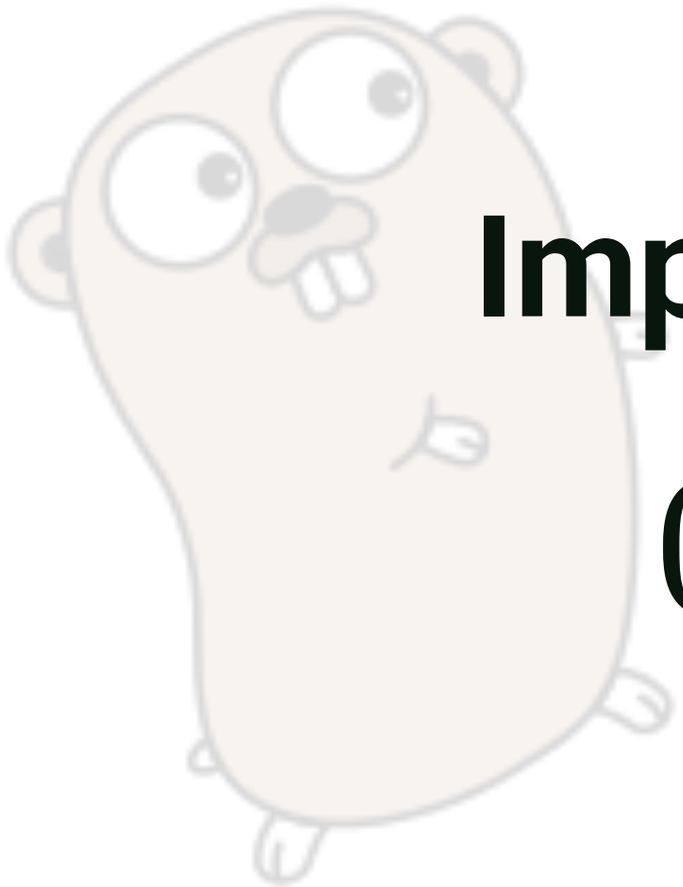
What’s wrong with inserted money?

Fix it!

Vending-machine

[x] after **adding coins** with **exact change** when **select item** the output should show Item without change


```go
func (m *VendingMachine) SelectSD() string {
    m.insertedMoney = 0
    return "SD"
}
func (m *VendingMachine) SelectCC() string {
    m.insertedMoney = 0
    return "CC"
}

func main() {
    …
    fmt.Println("Inserted Money:", vm.InsertedMoney())
    // Inserted Money: 17
    can = vm.SelectSD()
    fmt.Println(can) // SD
    vm.InsertCoin("T")
    vm.InsertCoin("TW")
    fmt.Println("Inserted Money:”, vm.InsertedMoney())
    // Inserted Money: 12
    can = vm.SelectCC()
    fmt.Println(can) // CC
}
```



Implement select DW (Drinking Water)

Moving on: Buy item without exact change

Vending-machine

[ ] if we **adding coins without exact change** when **select item** the **output should include change**


```go
func main() {
    …
    vm.InsertCoin(“T")
    vm.InsertCoin(“T")
    fmt.Println("Inserted Money:”, vm.InsertedMoney())
    // 20
    can = vm.SelectCC()
    fmt.Println(can) // CC, F, TW, O
}
```

Vending-machine

[ ] if we **adding coins without exact change** when **select item** the **output should include change**


```
func (m *VendingMachine) SelectCC() string {
    m.insertedMoney = 0
    return “CC” + “, F, TW, O”
}
func main() {
    …
    vm.InsertCoin(“T")
    vm.InsertCoin(“T")
    fmt.Println("Inserted Money:”, vm.InsertedMoney())
    // 20
    can = vm.SelectCC()
    fmt.Println(can) // CC, F, TW, O
}
```

As we know hard code is not the solution to solve this problem.

?

What can I do?

Let’s start with problem.

- We want to show coins that represent the change
- We want to know the change after select item
- We want to know the item price

Vending-machine

[ ] if we **adding coins without exact change** when **select item** the **output should include change**


```
func (m *VendingMachine) SelectCC() string {
    price := 12
    change := m.insertedMoney - price
    return "CC" + m.change(change)
}
func (m VendingMachine) change(c int) (change string) {
    return “, F, TW, O”
}
func main() {
    …
    vm.InsertCoin(“T")
    vm.InsertCoin(“T")
    fmt.Println("Inserted Money:”, vm.InsertedMoney())
    // 20
    can = vm.SelectCC()
    fmt.Println(can) // CC, F, TW, O
}
```

Things we have done.

[x] design method that represent the change

[x] the change after select item

[ ] want to know the item price.

How come the magic number “12”?

Vending-machine

[ ] if we **adding coins without exact change** when **select item** the **output should include change**


```
func (m *VendingMachine) SelectSD() string {
    price := 18
    change := m.insertedMoney - price
    return "SD" + m.change(change)
}
func (m *VendingMachine) SelectCC() string {
    price := 12
    change := m.insertedMoney - price
    return "CC" + m.change(change)
}
func (m VendingMachine) change(c int) string {
    if change == 0 {
        return “”
    }
    return “, F, TW, O”
}
```

Have you seen the pattern?

[Key : Value]

Vending-machine

[ ] if we **adding coins without exact change** when **select item** the **output should include change**


```
type VendingMachine struct {
    insertedMoney int
    coins         map[string]int
    items         map[string]int
}
func (m *VendingMachine) SelectSD() string {
    price := m.items[“SD”]
    change := m.insertedMoney - price
    return "SD" + m.change(change)
}
func (m *VendingMachine) SelectCC() string {
    price := m.items[“CC”]
    change := m.insertedMoney - price
    return "CC" + m.change(change)
}
func main() {
    var coins = map[string]int{"T": 10, "F": 5
                    , "TW": 2, "O": 1}
    var items = map[string]int{"SD": 18, "CC": 12}
    vm := VendingMachine{coins: coins, items: items}
}
```

Things we have done.

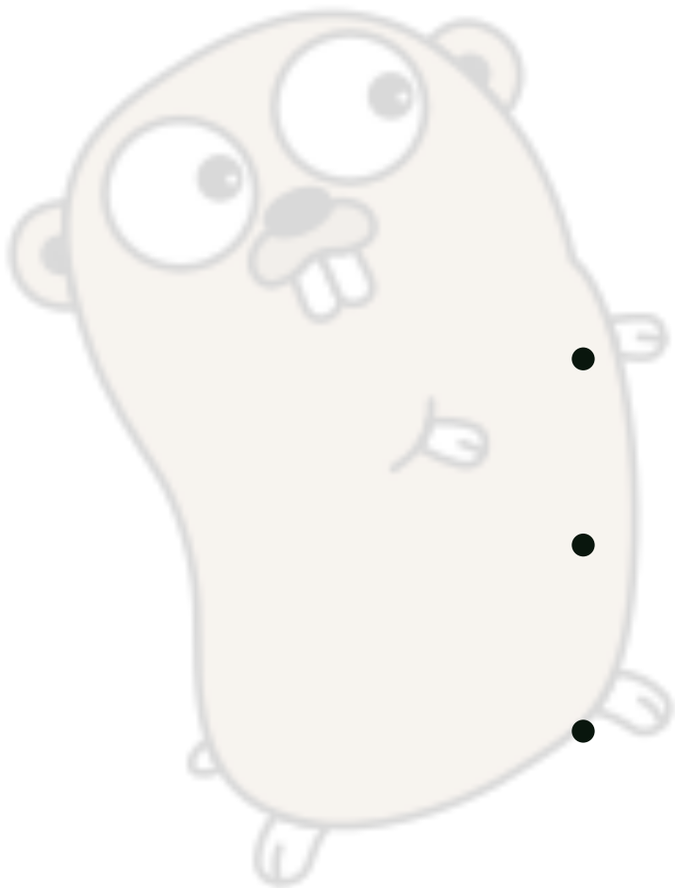
[x] design method that represent the change

[x] the change after select item

[x] want to know the item price.

How come the magic number “12”?

Implement Vending Machine Change method



- Adding coins: [T, T]
- Buy canned coffee: selectCC()
- Result should be “CC, F, TW, O”

```
return “SD" + m.change(change)
return "CC" + m.change(change)
```

Vending-machine

[ ] if we **adding coins without exact change** when **select item** the **output should include change**

c = 8, expected result = “, F, TW, O”

c = 0, expected result = “”

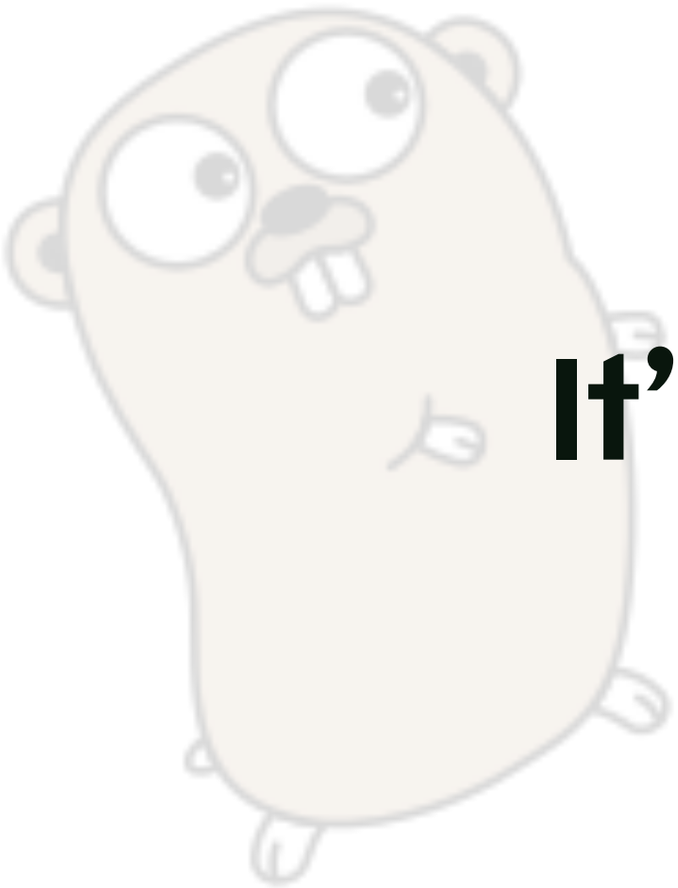

```
func (m *VendingMachine) change(c int) string {
    if c == 0 {
        return “”
    }
    return “, F, TW, O”
}
```

It’s work only case:

c = 8

c = 0

But!!!! Hard code



It’s time to refactor

Vending-machine

c = 8, expected result = “, F, TW, O”

c = 0, expected result = “”


```go
func (m *VendingMachine) change(c int) string {
    var str string
    if c == 8 {
        str += “, F, TW, O”
    }
    return str
}
```

Vending-machine

c = 8, expected result = “, F, TW, O”

c = 0, expected result = “”


```go
func (m *VendingMachine) change(c int) string {
    var str string
    if c >= 5 {
        str += “, F”
        c -= 5
    }
    if c >= 2 {
        str += “, TW”
        c -= 2
    }
    if c >= 1 {
        str += “, O”
        c -= 1
    }
    return str
}
```

Pattern?

Vending-machine

c = 8, expected result = “, F, TW, O”

c = 0, expected result = “”


```go
func (m *VendingMachine) change(c int) string {
    var str string
    if c >= 5 {
        str += “, F”
        c -= 5
    }
    if c >= 2 {
        str += “, TW”
        c -= 2
    }
    if c >= 1 {
        str += “, O”
        c -= 1
    }
    return str
}
```

Vending-machine

c = 8, expected result = “, F, TW, O”

c = 0, expected result = “”


```go
func (m *VendingMachine) change(c int) string {
    var str string
    if c >= 5 {
        str += “, F”
        c -= 5
    }
    if c >= 2 {
        str += “, TW”
        c -= 2
    }
    if c >= 1 {
        str += “, O”
        c -= 1
    }
    return str
}
```

Vending-machine

c = 8, expected result = “, F, TW, O”

c = 0, expected result = “”


```go
func (m *VendingMachine) change(c int) string {
    var str string
    values := [...]int{10, 5, 2, 1}
    if c >= values[0] {
        str += “, F”
        c -= values[0]
    }
    if c >= values[1] {
        str += “, TW”
        c -= values[1]
    }
    if c >= values[2] {
        str += “, O”
        c -= values[2]
    }
    return str
}
```

Vending-machine

c = 8, expected result = “, F, TW, O”

c = 0, expected result = “”


```go
func (m *VendingMachine) change(c int) string {
    var str string
    values := [...]int{10, 5, 2, 1}
    coins := [...]string{"T", "F", "TW", “O”}
    if c >= values[0] {
        str += “, ” + coins[0]
        c -= values[0]
    }
    if c >= values[1] {
        str += “, ” + coins[1]
        c -= values[1]
    }
    if c >= values[2] {
        str += “, ” + coins[2]
        c -= values[2]
    }
    return str
}
```

Vending-machine

c = 8, expected result = “, F, TW, O”

c = 0, expected result = “”


```
func (m *VendingMachine) change(c int) string {
    var str string
    values := [...]int{10, 5, 2, 1}
    coins := [...]string{"T", "F", "TW", “O”}

    for i := 0; i < len(values); i++ {
        if c >= values[i] {
          str += ", " + coins[i]
            c -= values[i]
        }
    }
    if c >= values[1] {
        str += “, ” + coins[1]
        c -= values[1]
    }
    if c >= values[2] {
        str += “, ” + coins[2]
        c -= values[2]
    }
    return str
}
```

Vending-machine

c = 8, expected result = “, F, TW, O”

c = 0, expected result = “”


```
func (m *VendingMachine) change(c int) string {
    var str string
    values := [...]int{10, 5, 2, 1}
    coins := [...]string{"T", "F", "TW", “O”}

    for i := 0; i < len(values); i++ {
        if c >= values[i] {
          str += ", " + coins[i]
            c -= values[i]
        }
    }
    return str
}
```

Vending-machine

[x] if we **adding coins without exact change** when **select item** the **output should include change**

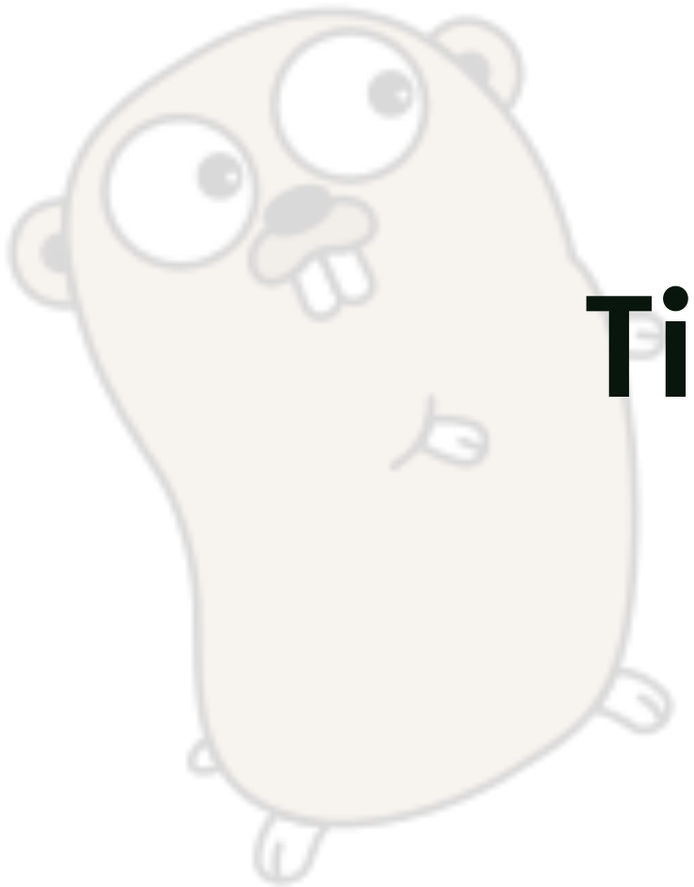

```
func (m *VendingMachine) change(c int) string {
    var str string
    values := [...]int{10, 5, 2, 1}
    coins := [...]string{"T", "F", "TW", “O”}

    for i := 0; i < len(values); i++ {
        if c >= values[i] {
          str += ", " + coins[i]
            c -= values[i]
        }
    }
    return str
}
```

Return coins

Vending-machine

[ ] if we **adding coins** and choose “**Return Coin**” then the change should **equals** the value that we have added


```go
func (m *VendingMachine) CoinReturn() string {
    return “T, T, F”
}
func main() {
    …
    vm.InsertCoin(“T")
    vm.InsertCoin(“T”)
    vm.InsertCoin(“F”)
    fmt.Println("Inserted Money:”, vm.InsertedMoney())
    // 25
    coin := vm.CoinReturn()
    fmt.Println(can) // T, T, F
}
```



Time to implement

Vending-machine

[ ] if we **adding coins** and choose “**Return Coin**” then the change should **equals** the value that we have added


```go
func (vm *VendingMachine) CoinReturn() string {
    coins := vm.change(vm.insertedMoney)
    vm.insertedMoney = 0
    return coins[2:len(coins)]
}
func (m *VendingMachine) change(c int) string {
    var str string
    values := [...]int{10, 5, 2, 1}
    coins := [...]string{"T", "F", "TW", “O”}

    for i := 0; i < len(values); i++ {
        if c >= values[i] {
          str += ", " + coins[i]
             c -= values[i]
             i--
        }
    }
    return str
}
```

Buy SD (Soft Drink)

and DW (Drinking Water)

without exact change