RIASSUNTO

`x[i:j]`
`x[:]`
} SUBLICING

`os.Args` } ARGOMENTI DA RIGA DI COMANDO

GENERAZIONE NUMERI PSEUDO-CASUALI

`r := rand.New(rand.NewSource( ))`

`r` → `*rand.Rand`

`time.Now().UnixNano()`

`r.Intn(50)` → restituisce un numero p. casuale tra 0, 1, ..., 49

```
func printDate(d Data) {
    fmt.Println(d.g, "/", d.m, "/",
                d.a)
}
```

3/1/2023 03/01/2023

fmt.Printf

fmt.Printf(STRING 4 DI FORMATO , 2 tri arg, ...)

n° argomenti

VERBI (% _)

abc %a %d ecc %3f ....

n° verbi =

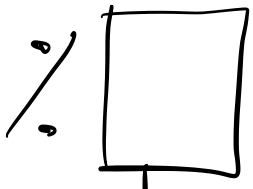
%d intero

%f float

%s string

%□

%□ verbo



↓

avverbio

```
func printDate(d Data) {
    fmt.Printf("%d/%d/%d\n",
        d.g, d.m, d.a)
}

fmt.Printf("Milano, il %d del mese
    %d dell'anno %d \n",
    d.g, d.m, d.a)
```

%4d

AMPIEZZA (almeno 4 caratteri)
padding a sx con spazi

fmt.Printf("%2d/%2d/%4d\n", d.g, d.m, d.a)

1/3/2023

%7.2f

AMPIEZZA = ESATTAMENTE LA VIRGOLA

CARATTERI DOPO 2

x := 1347.43162945

fmt.Printf("%9.2f\n", x)

␣␣1347.43

%02d

%2d ma padding con "0"

```
func printDate(d Data) {
    fmt.Printf("%02d/%02d/%04d\n",
        d.g, d.m, d.a)
}

func data2string(d Data) string {
    return fmt.Sprintf("%02d/
        %02d/%04d", d.g, d.m, d.a)
}
```

ESERCIZIO

type Persona struct {
nome, cognome string
nascita data
cf string
}

Scrivete una funzione che data una slice di persone e un cognome, restituisce quante persone hanno quel cognome.

```
func conta(p []Persona, c string) int {
    var conta int
    for _, persona := range p {
        if persona.cognome == c {
            conta++
        }
    }
    return conta
}
```

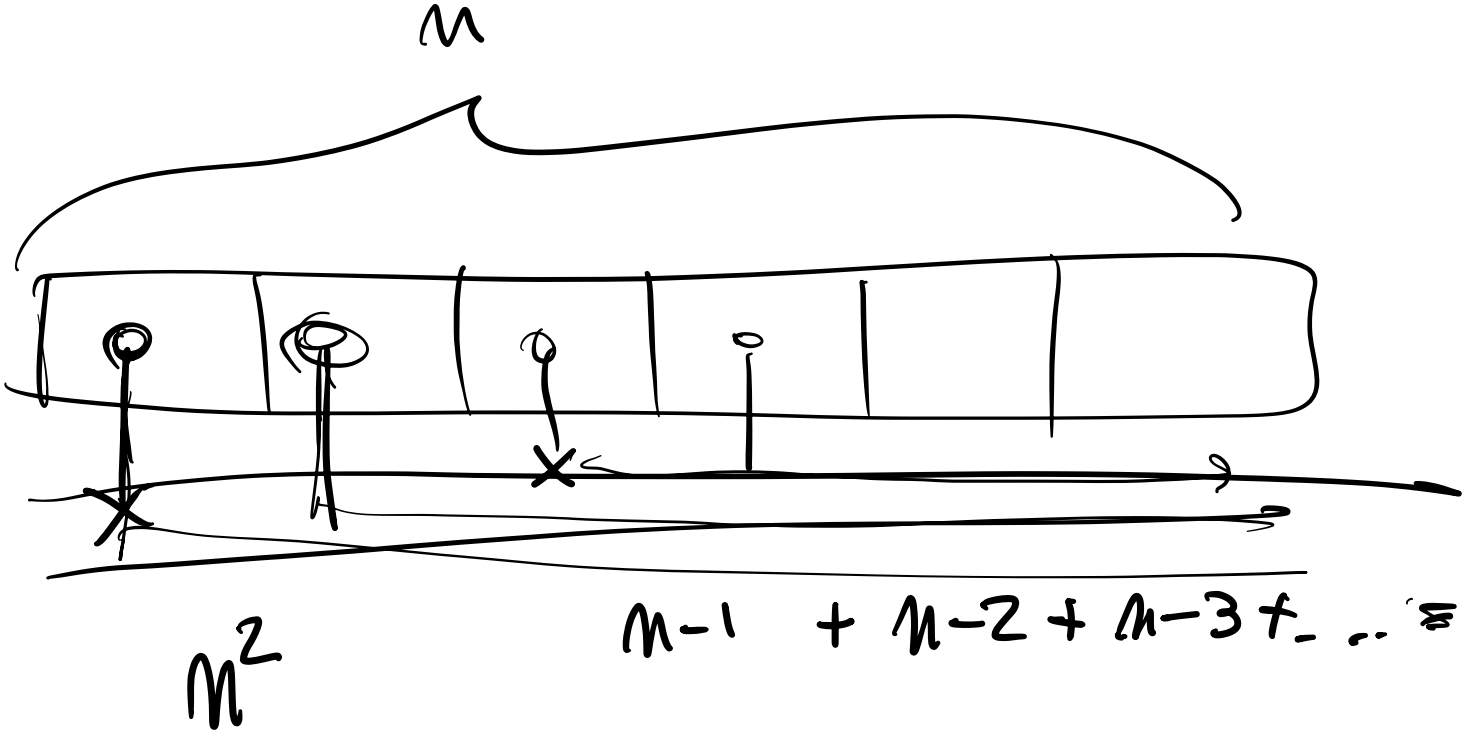
[]Persona

```
func selez (p []Persona, c string) {
    var risultato []Persona
    for _, persona := range p {
        if persona.cognome == c {
            risultato = append(risultato,
                persona)
        }
    }
    return risultato
}
```

ESERCIZIO

type Persona struct {
nome, cognome string
nascita data
cf string
}

Scrivete una funzione che data una slice di persone guarda se ci sono codici fiscali duplicati.

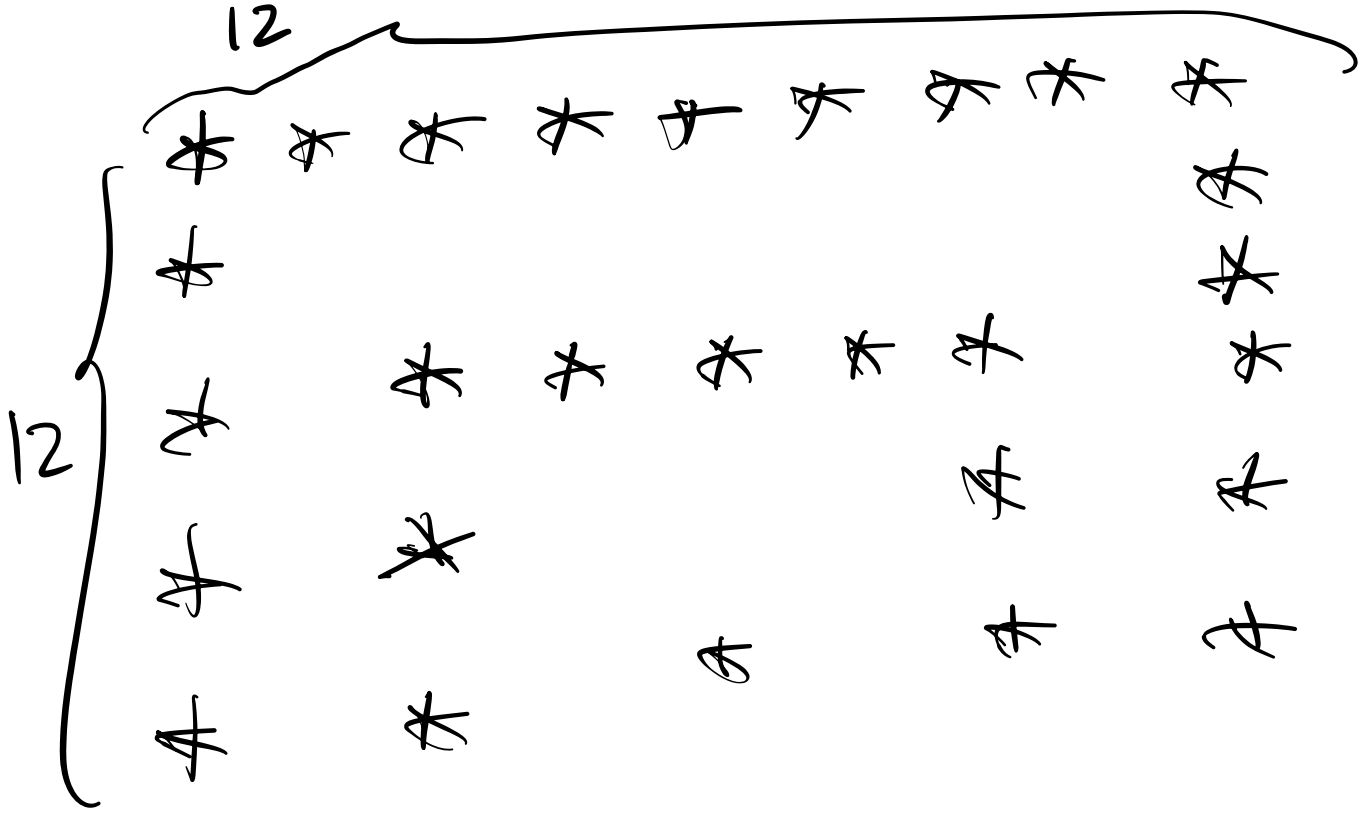


```
func containsDuplicates(p []Persone) bool {
    for i:=0; i<len(p); i++ {
        for j:=i+1; j<len(p); j++ {
            if p[i].cf == p[j].cf {
                return true
            }
        }
    }
    return false
}
```

ESERCIZIO

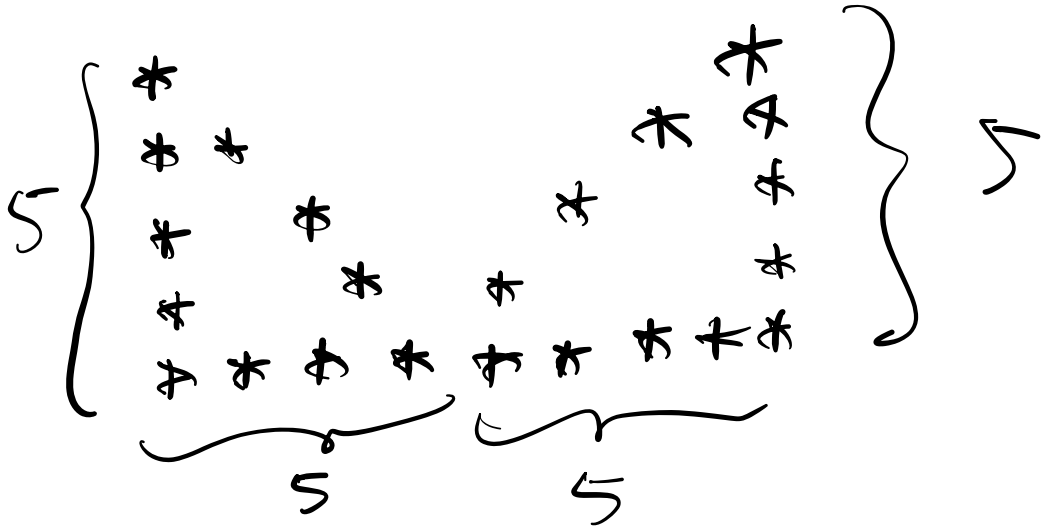
Scrivere un programma

```
$>./concentrico 12
```

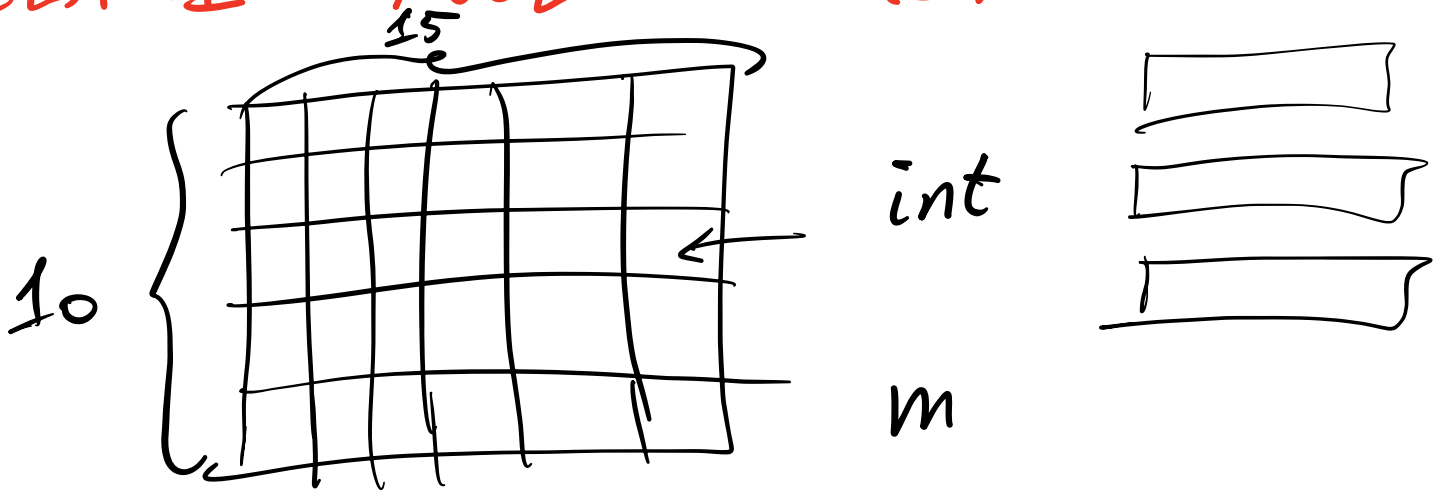


ESERCIZIO

piramidi 5



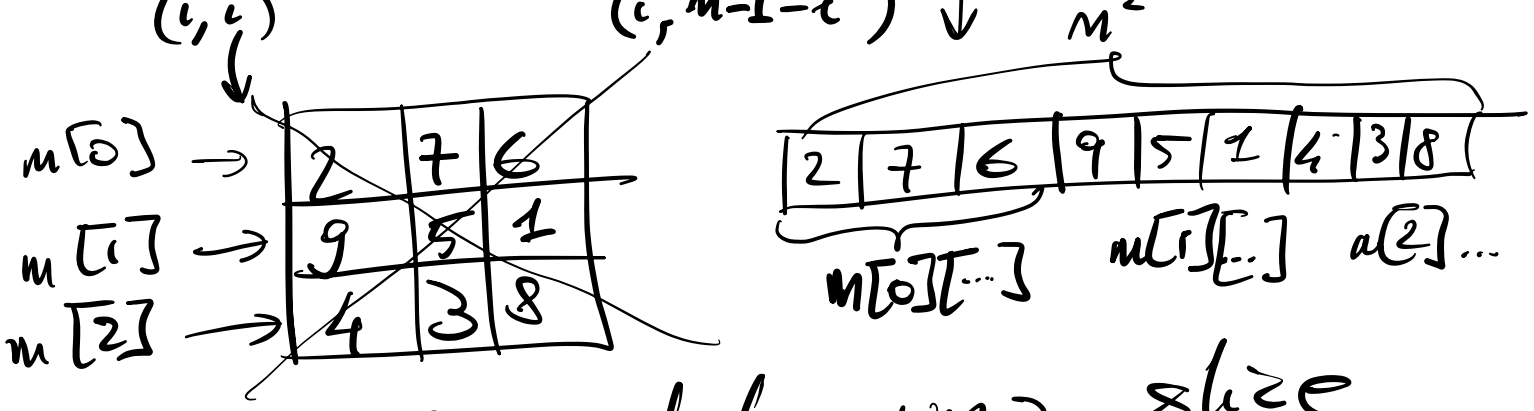
SLICE MULTIDIMENSIONALI




```
var m [][]int
m = make([][]int, 10)
for r := 0; r < 10; r++ {
    m[r] = make([]int, 15)
}
```

`M[i][j]`

`z[k] = m[k/n][k%n]`

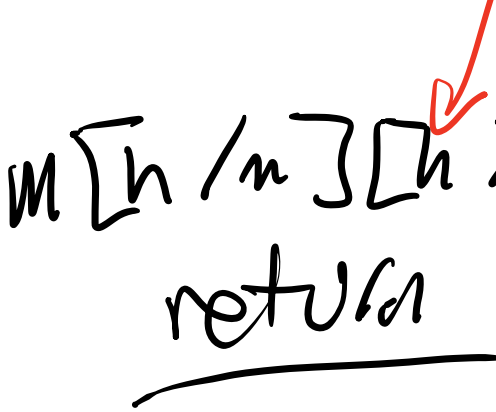


Funzione che data una slice quadrata di interi stabilisce se è un quadrato magico

```
func isMagic (m [][]int) bool {
    n := len(m)
    var primaRiga
    for j := 0; j < n; j++ {
        primaRiga += m[0][j]
    }
    //Controllo le altre righe
    for i := 1; i < n; i++ {
        s := 0
        for j := 0; j < n; j++ {
            s += m[i][j]
        }
```

```
        if s != primaRiga {
            return false
        }
    }
} // Controlla le colonne
for j := 0; j < n; j++ {
    s := 0
    for i := 0; i < n; i++ {
        s += m[i][j]
    }
    if s != primaRiga {
        return false
    }
}
// Controllo le diagonali
s := 0
for i := 0; i < n; i++ {
    s += m[i][i]
}
```

```
if  s != prima Rija {
        return false
}
s = 0
for i := 0; i < n; i++ {
    s += m[i][n-i-1]
}
if  s != prima Rija {
        return false
}
}
for k := 0; k < n*n; k++ {
    for h := k+1; h < n*n; h++ {
        if z[h] == z[k] {
            return false
        }
    }
}
}
```



```
M[h/n][h%n] == M[R/n][R%n]
    return true
}
```

```
type Carta struct {
    seme  int  //0=cori, 1=quadri, ...
    valore int  //0=A, 1=2, ..., 9=10,
                //10=J, 11=Q, 12=K
}

func mazzo() []Carte {
    var m []Carte
    for s:=0; s<4; s++ {
        for v:=0; v<13; v++ {
            var c Carta
            c.seme = s
            c.valore = v
            m = append(m, c)
        }
    }
    return m
}
```