مفهوم آرایه

فرض کن می‌خوای نمرات 3 درس یه دانش‌آموزو توی برنامه ذخیره کنی (تمرین شماره 2 درسنامه "مقایسه و شرط") خب شاید اینطوری بنویسی:

```
var math float32 = 14.0
var farsi float32 = 18.0
var physicalEducation float32 = 17.0
```

حالا اگه بخوای نمره 12 تا درس یه دانش‌آموزو ذخیره کنی چیکار میکنی؟ میای 12 تا متغیر در نظر میگیری برای ذخیره کردن نمرات؟

اینجاست که باید از آرایه استفاده کنیم

به طور کلی هر موقع قصد انجام محاسبات و عملیات روی تعداد متغیر از یک نوع (که از نظر معنایی هم تقریبا با هم در ارتباط هستند) داشته باشیم بهتره از متغیر استفاده کنیم. این کار باعث میشه انجام عملیات و محاسبات روی اونها با سهولت بیشتری انجام بشه.

آرایه

آرایه ساختاری است که می‌تونه چند تا مقدار از یه نوع مشخص رو کنار هم نگه‌داری کنه.

یعنی یه جعبه‌ی مرتب‌شده‌ست، که توش مثلاً فقط عدد، یا فقط رشته، یا فقط بولین می‌تونی بذاری.

روش تعریف آرایه با طول ثابت

```
var <name> [<length>]<type>
```

var

کلمه‌ی کلیدی برای تعریف متغیر

name

نام متغیر برای دسترسی به اون

یعنی قراره با این اسم به آرایه‌مون دسترسی داشته باشیم.

هر اسمی می‌تونی بذاری (مثل grades)

length

تعداد عناصر آرایه که حتما باید یک عدد صحیح بزرگتر از 0 باشه

به عنوان مثال اگه این عدد رو 3 قرار بدیم یعنی یک آرایه داریم به طول 3

این آرایه طولش ثابت هست و نمیتونی بیشتر از طولی که براش مشخص کردی عنصر داخلش قرار بدی

type

نوع داده عناصر آرایه

این میتونه یکی از انواع مجاز زبان GO (مثل int یا float32 و ...) باشه

تمام عناصر آرایه باید از نوع همین type باشن

مثلا اگه float32 باشه تمام عناصر آرایه میتونن عدد اعشاری باشن

مثال

```
var grades [3]float32
```

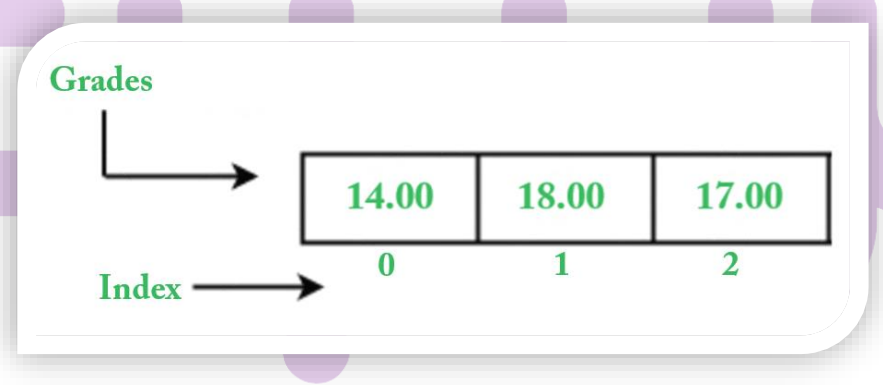
در اینجا یک آرایه به نام grades تعریف کردیم که میتونه 3 عنصر از نوع float32 نگهداری کنه

ایندکس در آرایه

وقتی آرایه رو تعریف میکنیم یک طول خاص براش مشخص میکنیم به عنوان مثال 3

در این صورت ما 3 عنصر در اون آرایه داریم. در واقع برای دسترسی به هر عنصر باید ترتیبِ اون عنصر به همراه نام اون آرایه رو کنار هم قرار بدیم

به تصویر زیر توجه کن



برای دسترسی به اولین عنصر آرایه داریم:

```
grades[0]
```

برای دسترسی به دومین عنصر آرایه داریم:

```
grades[1]
```

برای دسترسی به سومین عنصر آرایه داریم:

```
grades[2]
```

وقتی نام متغیر یک آرایه رو در کنار اپراتور [x] (که x میتونه یک عدد صحیح بزرگتر/مساوی 0 و کوچیکتر از طول آرایه باشه) مینویسیم در واقع داریم از عملگر ایندکس استفاده میکنیم. به طور خلاصه عملگر ایندکس باعث میشه بتونیم به یک عنصر خاص از آرایه دسترسی داشته باشیم و عملیات خوندن و نوشتن بر اون عنصر انجام بدیم

روش دسترسی به عناصر آرایه

حالا که با مفهوم ایندکس در آرایه آشنا شدیم میتونیم به عناصر آرایه دسترسی داشته باشیم و عملیات خوندن و نوشتن بر هر کدوم از عنصرهاش انجام بدیم

وقتی که آرایه رو برای اولین بار تعریف میکنیم و هنوز هیچ مقداری به عناصر اون آرایه ندادیم مقدار هر کدوم از عنصرهاش همون مقدار اولیه نوع اون آرایه هست. در این مثالی که در حال بررسیش هستیم از اونجایی که نوع آرایه float32 بوده بنابراین مقدار اولیه هر عنصر 0.0 هست.

اگه هر کدوم از عنصرهای این آرایه (یعنی عنصر اول، دوم و سوم) رو با استفاده از دستور چاپ در خروجی چاپ کنیم به صحت این موضوع پی میبریم

```
package main

import "fmt"

func main() {
    var grades [3]float32

    fmt.Println("عنصر اول آرایه نمرات: ", grades[0])
    fmt.Println("عنصر دوم آرایه نمرات: ", grades[1])
    fmt.Println("عنصر سوم آرایه نمرات: ", grades[2])
}
```

خروجی این برنامه در زیر آورده شده

```
عنصر اول آرایه نمرات: 0.0
عنصر دوم آرایه نمرات: 0.0
عنصر سوم آرایه نمرات: 0.0
```

در واقع در این مثال مقدار ذخیره شده هر کدام از عنصرهای آرایه رو خوندیم

حالا قصد داریم مقدارهای ذخیره شده (مقدار اولیه) در عناصر آرایه رو تغییر بدیم

این کار با استفاده از همون عملگر ایندکس انجام میشه

```
package main

import "fmt"

func main() {
    var grades [3]float32
    grades[0] = 14.0
    grades[1] = 18.0
    grades[2] = 17.0

    fmt.Println("عنصر اول آرایه نمرات:", grades[0])
    fmt.Println("عنصر دوم آرایه نمرات:", grades[1])
    fmt.Println("عنصر سوم آرایه نمرات:", grades[2])
}
```

خروجی این برنامه در زیر آورده شده

```
عنصر اول آرایه نمرات: 14.0
عنصر دوم آرایه نمرات: 18.0
عنصر سوم آرایه نمرات: 17.0
```

دسترسی خارج از محدوده

به طور کلی ما مجاز هستیم با استفاده از عملگر اینکدس تنها در بازه مجاز (از ایندکس 0 تا ایندکس LENGTH – 1) دسترسی داشته باشیم. در صورتی که به یک عنصر خارج از محدوده مورد نظر دسترسی داشته باشیم **در هنگام اجرا** خطا دریافت میکنیم

به مثال زیر توجه کن. در این مثال به اشتباه برای دسترسی به عنصر سوم به جای استفاده از عملگر ایندکس [2] از عملگر ایندکس [3] استفاده کردیم

```
package main


import "fmt"


func main() {
    var grades [3]float32
    grades[0] = 14.0
    grades[1] = 18.0
    grades[2] = 17.0


    fmt.Println("عنصر اول آرایه نمرات: ", grades[0])
    fmt.Println("عنصر دوم آرایه نمرات: ", grades[1])
    fmt.Println("عنصر سوم آرایه نمرات: ", grades[3])
}
```

خروجی

```
panic: runtime error: index out of range [3] with length 3
```

نکته: باید توجه داشته باشیم که در این صورت در هنگام اجرا خطا دریافت میکنیم و این خطا کمی خطرناک تر از خطای کامپایل هست. در زیر تفاوت خطای کامپایل و اجرا به واضح توضیح داده شده است.

تفاوت خطای کامپایل (Compile-time Error) و خطای اجرا(Runtime Error)

خطای کامپایل (Compile-time Error)

این خطا زمانی اتفاق می‌افته که هنوز **برنامه اجرا نشده** و فقط کامپایلر (مفسر زبان GO) داره کدتو بررسی می‌کنه.

یعنی قبل از اینکه حتی یه خط از برنامه اجرا بشه، کامپایلر می‌فهمه یه چیزایی تو کدت مشکل داره و نمی‌ذاره برنامه ساخته بشه.

چند مثال از خطاهای کامپایل

```
var name int = "Ali" // ✗ نوع مقدار اشتباهه (string رو ریختی تو int)
```

```
fmt.Println(score // ✗ پرانتز بسته نشده
```

```
undeclaredVar = 10 // ✗ متغیر تعریف نشده
```

خطای اجرا (Runtime Error)

این خطا **بعد از کامپایل موفق** و **زمانی که برنامه در حال اجراست** اتفاق می‌افته.

یعنی:

کامپایلر فکر می‌کنه همه‌چی اوکیه، ولی وقتی برنامه در عمل اجرا می‌شه، به مشکل می‌خوره.

چند مثال از خطاهای اجرا

```
var grades [3]int

fmt.Println(grades[5]) // ✗ خطای زمان اجرا: ایندکس خارج از محدوده
```

```
var x int

y := 10 / x // ✗ تقسیم بر صفر (در زمان اجرا panic)
```

چون:

1. **تا برنامه اجرا نشه، متوجه‌ش نمی‌شی!**
   یعنی ممکنه برنامه ظاهراً سالم باشه، ولی یهو وسط اجرای واقعی کرش کنه.
2. **توی برنامه‌های واقعی می‌تونه به داده یا کاربر آسیب بزنه.**
   مثلاً یه برنامه بانکی که یه عدد اشتباه رو محاسبه کنه ولی خطایی نده !
3. **پیدا کردنشون سخت‌تره.**
   چون ممکنه فقط در شرایط خاص (ورودی خاص، زمان خاص) خودشونو نشون بدن.
4. **کاربر نهایی ممکنه باهاش مواجه بشه.**
   ولی خطای کامپایل معمولاً تو مرحله‌ی توسعه دیده میشه.

تعریف و مقداردهی همزمان آرایه

اگه در هنگام تعریف آرایه بدونیم چه مقادیری میخوایم داخلش قرار بدیم میتونیم به طور همزمان هم تعریفش کنیم و هم مقدار دهی کنیم، این کار باعث میشه کدها کمتر بشن و همچینین باعث خوانایی بیشتر برنامه میشه.

به روش زیر میتونیم یک آرایه رو تعریف و مقدار دهی کنیم

```
var <name> [<length>]<type> = [<length>]<type>{value1, value2, value3, …, valueN}
```

کلمه‌ی کلیدی برای تعریف متغیر

name

نام متغیر برای دسترسی به اون

یعنی قراره با این اسم به آرایه‌مون دسترسی داشته باشیم.

هر اسمی می‌تونی بذاری (مثل grades)

length

تعداد عناصر آرایه که حتما باید یک عدد صحیح بزرگتر از 0 باشه

به عنوان مثال اگه این عدد رو 3 قرار بدیم یعنی یک آرایه داریم به طولی 3

این آرایه طولش ثابت هست و نمیتونی بیشتر از طولی که براش مشخص کردی عنصر داخلش قرار بدی

type

نوع داده عناصر آرایه

این میتونه یکی از انواع مجاز زبان GO (مثل int یا float32 و ...) باشه

تمام عناصر آرایه باید از نوع همین type باشن

مثلا اگه float32 باشه تمام عناصر آرایه میتونن عدد اعشاری باشن

= [<length>]<type>{...}

این قسمت برای مقداردهی اولیه به خونه‌های آرایه‌ست.
تو این بخش باید به ترتیب، مقادیر موردنظرتو بنویسی.

به مثال زیر توجه کن

```
package main


import "fmt"


func main() {
    var grades [3]float32 = [3]float32 {14.0, 18.0, 17.0}


    fmt.Println("عنصر اول آرایه نمرات: ", grades[0])
    fmt.Println("عنصر دوم آرایه نمرات: ", grades[1])
    fmt.Println("عنصر سوم آرایه نمرات: ", grades[3])
}
```

همونطور که مشخصه تو این حالت دیگه نیاز نیست هر موقعیت از آرایه را به صورت جدا مقداردهی کنیم و این باعث شد که 3 خط از تعداد کدها کم بشه

تو این حالت میشه تعریف و مقداردهی آرایه رو کمی خلاصه تر هم کرد. یادت میاد تو حالتی که یک متغیر تعریف میکردیم و درجا مقداردهی میکردیم؟ تو اون حالت دیگه نیاز نبود نوع اون متغیر رو بنویسیم. در واقع کامپایلر خودش بر اساس اون مقدار دهی که کرده بودیم متوجه میشد نوع متغیر چی هست. در اینجا هم میشه این کارو انجام داد

در برنامه زیر این ساده سازی انجام شده

```
package main

import "fmt"

func main() {
    var grades = [3]float32 {14.0, 18.0, 17.0}

    fmt.Println("عنصر اول آرایه نمرات: ", grades[0])
    fmt.Println("عنصر دوم آرایه نمرات: ", grades[1])
    fmt.Println("عنصر سوم آرایه نمرات: ", grades[3])
}
```

ما حتی میتونیم از این فراتر هم بریم. از اونجایی که مقداردهی اولیه داره دقیقا تو همون خطی که آرایه رو تعریف میکنیم انجام میشه نیاز نیست تعداد عنصرهارو در نوع بنویسیم. چون با توجه به اینکه 3 مقدار در بخش مقداردهی نوشتیم مشخصه که طول آرایه 3 هستش، کافیه به جای عدد 3 (که معادل طول آرایه هست) ... قرار بدیم. این باعث میشه کامپایلر خودش سعی کنه با توجه به تعداد مقادیر تعداد طول آرایه رو محاسبه کنه.

در برنامه زیر این ساده سازی انجام شده

```
package main

import "fmt"

func main() {
    var grades = […]float32 {14.0, 18.0, 17.0}

    fmt.Println("عنصر اول آرایه نمرات: ", grades[0])
    fmt.Println("عنصر دوم آرایه نمرات: ", grades[1])
    fmt.Println("عنصر سوم آرایه نمرات: ", grades[3])
}
```

تا به الان هر مقداردهی که انجام دادیم به ترتیب انجام شده. یعنی:

عدد 14.00 در اولین مکان حافظه آرایه قرار گرفته و با ایندکس 0 در دسترس هست

عدد 18.00 در دومین مکان حافظه آرایه قرار گرفته و با ایندکس 1 در دسترس هست

عدد 17.00 در سومین مکان حافظه آرایه قرار گرفته و با ایندکس 2 در دسترس هست

ما میتونیم درهنگام مقداردهی اولیه ترتیب رو خودمون مشخص کنیم.

به مثال زیر توجه کن

```
package main

import "fmt"

func main() {
    var grades = […]float32 {2: 14.0, 1: 18.0, 0: 17.0}

    fmt.Println(“عنصر اول آرایه نمرات:”, grades[0])
    fmt.Println(“عنصر دوم آرایه نمرات:”, grades[1])
    fmt.Println(“عنصر سوم آرایه نمرات:”, grades[3])
}
```

تو این حالت عناصر به شکل زیر در آرایه قرار گرفتند

عدد 14.00 در سومین مکان حافظه آرایه قرار گرفته و با ایندکس 2 در دسترس هست

عدد 18.00 در دومین مکان حافظه آرایه قرار گرفته و با ایندکس 1 در دسترس هست

عدد 17.00 در اولین مکان حافظه آرایه قرار گرفته و با ایندکس 0 در دسترس هست

در این حالت حتی میتونیم تنها به بعضی از موقعیت های آرایه رو مقداردهی کنیم و اجازه بدیم باقی موقعیت های از مقدار zero value استفاده کنند. فقط باید حواسمون باشه که در این صورت حتما باید طول آرایه رو مشخص کنیم.

به مثال زیر توجه کن

```
package main


import "fmt"


func main() {
    var familyMembers = [5]float32 {3: “neda”, 4: “arian”}


    fmt.Println(“اولین عضو خانواده:”, familyMembers [0])
    fmt.Println(“دومین عضو خانواده:”, familyMembers [1])
    fmt.Println(“سومین عضو خانواده:”, familyMembers [2])
    fmt.Println(“چهارمین عضو خانواده:”, familyMembers [3])
    fmt.Println(“پنجمین عضو خانواده:”, familyMembers [4])
}
```

گرفتن مقدار از ورودی

برنامه ای که برای تبدیل درجه سانتیگراد به فارنهایت نوشتیم یادت میاد؟

```
package main

import "fmt"

func main() {
    fmt.Println("دما به فارنهایت:", (30.0 * 1.8) + 32)
}
```

اگه این برنامه رو کامپایل و اجرا کنیم و همواره داره دمای 30.0 درجه سانتیگراد رو به فارنهایت تبدیل میکنه.

حالا اگه بخوایم به جای 30.0 درجه سانتیگراد بیایم 40.0 درجه سانتیگراد رو به فارنهایت تبدیل کنیم چیکار باید بکنم؟

یک راه ساده اینه که بیایم تو سورس کد برنامه به جای عدد 30.0 عدد 40.0 قرار بدیم و برنامه رو دوباره کامپایل واجرا کنیم. اما اگه بخوایم این برنامه رو به یه فردی بدیم که هیچ دانش برنامه نویسی نداره چی؟ اون چجوری میخواد بیاد این تغییر رو انجام بده و مجدد برنامه رو کامپایل و اجرا کنه! حتی اگه فرض کنیم برنامه نویسی هم بلده چرا باید اینکارو انجام بدم؟ کامپایل کردن برنامه نیاز به نصب بودن کامپایلر روی اون دستگاه داره همچنین کامپایل برنامه ها همیشه اینقدر سریع انجام نمیشه گاهی ممکنه برنامه بزرگ باشه و کامپایل کردنش دقیقه ها طول بکشه! این منطقی نیست که برای این کار ساده مجبور شیم برنامه رو از اول کامپایل کنیم

روش بهتر اینه که در هنگام برنامه نویسی اینجور در نظر بگیریم که مقدار دما به سانیتگراد قراره توسط کسی که برنامه رو اجرا میکنه وارد بشه و برنامه باید درست پیش از اینکه محاسبه رو انجام بده منتظر بمونه تا کاربر مقدار دما رو وارد کنه. این باعث میشه برنامه انعطاف پذیر بشه و برای هر مقادیر مختلف محاسبه رو انجام بده

در زبان GO برای گرفتن ورودی از کاربر به شکل زیر عمل میکنیم

```
package main


import "fmt"


func main() {
    var c float32
    fmt.Println("دمایی که میخوای از سانتیگراد به فارنهایت تبدیل بشه وارد کن")
    fmt.Scanln(&c)
    fmt.Println("دما به فارنهایت:", (c * 1.8) + 32)
}
```

"دمایی که می‌خوای از سانتی‌گراد به فارنهایت تبدیل بشه رو وارد کن":

بعد منتظر می‌مونه تا کاربر یه مقدار وارد کنه.

به محض اینکه کاربر مقدار موردنظر رو تایپ کرد و دکمه‌ی Enter رو فشار داد، برنامه به خط بعدی می‌ره و محاسبه رو انجام می‌ده.

در زمان گرفتن ورودی از کاربر ممکنه خطا اتفاق بیفته.
مثلاً تو این مثال ما انتظار داریم کاربر یه عدد اعشاری وارد کنه.
اما اگه کاربر به‌جای عدد، یه متن مثل "hello" وارد کنه، برنامه موقع اجرا (runtime) با خطا مواجه می‌شه و ممکنه متوقف بشه.

عملگر &

وقتی از fmt.Scanln() استفاده می‌کنیم، باید **آدرس متغیر** رو بهش بدیم.
برای این کار از **عملگر &** استفاده می‌کنیم.
این عملگر، آدرس حافظه‌ی متغیر رو به تابع می‌ده تا مقدار ورودی رو مستقیماً داخل همون متغیر ذخیره کنه.

اگه & رو ننویسیم و فقط بنویسیم Scanln(temp)، برنامه درست کار نمی‌کنه چون داره مقدار متغیر رو می‌فرسته، نه آدرسش رو.

عملگرهای مجاز در آرایه با طول ثابت

عملگر ==

دو آرایه به شرطی که طول آن ها یکسان باشد و از نظر نوع همسان باشند می توانند توسط عملگر == مقایسه شوند. در صورتی که هر دو آرایه نظیر به نظیر مقادیر یکسان داشته باشند حاصل این عملگر true است در غیر این صورت false

```
package main

import "fmt"

func main() {
    var arianGrades [3]float32 = [3]float32 {14.0, 18.0, 17.0}
    var nargesGrades [3]float32 = [3]float32 {14.0, 18.0, 17.0}

    fmt.Println("نمرات آرین و نرگس مثل هم هست: ", arianGrades == nargesGrades)
}
```

عملگر !=

دو آرایه به شرطی که طول آن ها یکسان باشد و از نظر نوع همسان باشند می توانند توسط عملگر != مقایسه شوند. در صورتی که دست کم یکی از عناصر دو آرایه نظیر به نظیر با هم تفاوت داشته باشد حاصل این عملگر true است در غیر اینصورت false

```
package main

import "fmt"

func main() {
    var arianGrades [3]float32 = [3]float32 {14.0, 18.0, 17.0}
    var omidGrades [3]float32 = [3]float32 {6.0, 20.0, 19.0}

    fmt.Println("نمرات آرین و امید مثل هم نیست: ", arianGrades != omidGrades)
}
```

عملگر انتساب =

اگر آرایه A هم‌طول و هم نوع آرایه B باشد می توان آن را توسط عملگر = به B متنسب کرد و برعکس. این عملگر سبب می شود آرایه سمت چپ عملگر همانند آرایه سمت راست عملگر شود.

```
package main


import "fmt"


func main() {
    var arianGrades [3]float32 = [3]float32 {14.0, 18.0, 17.0}
    var nargesGrades [3]float32 = arianGrades
    fmt.Println(“نمرات نرگس:”, nargesGrades)
}
```

آرایه های تو در تو با طول ثابت

فرض کن می‌خوای اطلاعات مربوط به نمرات چند دانش‌آموزو تو برنامه ذخیره کنی. مثلاً نمرات 3 درس برای 4 دانش‌آموز:

	ریاضی	فارسی	تربیت بدنی
آرین	14.0	18.0	17.0
ندا	18.0	19.5	16.5
امید	6.0	20.0	19.0
نیما	18.0	17.75	18.0

اگه بخوای فقط نمرات یک دانش‌آموز رو ذخیره کنی، یه آرایه ساده مثل این کافیه:

```
var arianGrades [3]float32 = [3]float32 {14.0, 18.0, 17.0}
```

اما وقتی برای **چند نفر** بخوای همچین داده‌ای داشته باشی، باید یه آرایه داشته باشی که **خودش شامل آرایه‌های دیگه‌ست**. یعنی **آرایه تو در تو**.

روش تعریف آرایه تو در تو با طول ثابت

```
var <name> [<rows_length>][<columns_length>]<type>
```

var

کلمه‌ی کلیدی برای تعریف متغیر

name

نام متغیر برای دسترسی به اون

یعنی قراره با این اسم به آرایه‌مون دسترسی داشته باشیم.

هر اسمی می‌تونی بذاری (مثل grades)

rows_length

تعداد سطرهای آرایه که که حتما باید یک عدد صحیح بزرگتر از 0 باشه

به عنوان مثال اگه این عدد رو 3 قرار بدیم یعنی یک آرایه داریم با 3 سطر که هر سطر خودش یک آرایه جداگانه با ستون های مشخصه

این تعداد ثابت هست و نمیتونی بیشتر از اون سطر ایجاد کنی

columns_length

تعداد ستون های هر سطر در آرایه که حتما باید یک عدد صحیح بزرگتر از 0 باشه

به عنوان مثال اگه این عدد رو 3 قرار بدیم یعنی هر سطر 3 ستون (یا 3 مقدار) داره

این آرایه طولش ثابت هست و نمیتونی بیشتر از طولی که براش مشخص کردی عنصر داخلش قرار بدی

type

نوع داده عناصر آرایه

این میتونه یکی از انواع مجاز زبان GO (مثل int یا float32 و ...) باشه

تمام عناصر آرایه باید از نوع همین type باشن

مثلا اگه float32 باشه تمام عناصر آرایه میتونن عدد اعشاری باشن

```
var allGrades [4][3]float32
```

در اینجا یک آرایه به نام allGrades تعریف کردیم که 4 سطر داره که هر سطرش میتونه 3 عنصر از نوع float32 نگهداری کنه

تمام حالت هایی که برای تعریف و مقداردهی همزمان در آرایه با طول ثابت داشتیم در اینجا نیز برقرار هست

به عنوان مثال میتونیم طبق جدول به صورت زیر مقداردهی انجام دهیم

```
var allGrades [4][3]float32 = [4][3]float32{
     [3]float32{14.0, 18.0, 17.0},
     [3]float32{18.0, 19.5, 16.5},
     [3]float32{6.0, 20.0, 19.0},
     [3]float32{18.0, 17.75, 18.0},
}
```

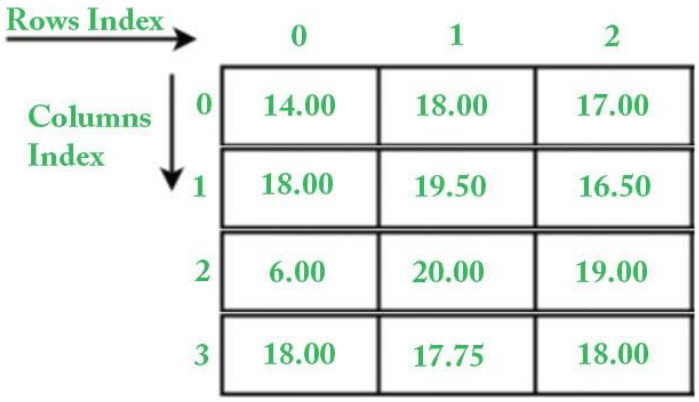
حتی این مقداردهی رو میشه کمی ساده تر کرد. از اونجایی که در نوع آرایه مشخص کردیم هر سطر خودش یک آرایه از نوع float32 به طول 3 هستش در هنگام مقداردهی نیاز نیست این نوع رو مجدد تکرار کنیم

```
var allGrades = [4][3]float32{
     {14.0, 18.0, 17.0},
     {18.0, 19.5, 16.5},
     {6.0, 20.0, 19.0},
     {18.0, 17.75, 18.0},
}
```

نکته:

آرایه‌های تو در تو **می‌تونن بیشتر از 2 بعدی هم باشن**، مثلاً 3 بعدی یا حتی 4 بعدی.
اما این نوع آرایه‌ها معمولاً **خیلی کاربرد خاص دارن** و در موارد عمومی یا روزمره زیاد استفاده نمی‌شن.

ما اینجا فقط راجع‌به **آرایه‌های 2 بعدی** صحبت کردیم چون **پرکاربردترین نوع آرایه‌ی تو در تو هستن**، مخصوصاً وقتی با داده‌های جدولی یا ماتریس‌ها کار می‌کنیم.



چطوری داده‌ها رو بخونیم یا بهشون دسترسی داشته باشیم؟

برای رسیدن به یک عدد، باید **ایندکس ردیف** و **ایندکس ستون** رو داشته باشیم.

مثلا allGrades[1][2]: یعنی عدد ردیف ۱ و ستون ۲ (یادت باشه اندیس‌ها از صفر شروع میشن!)

مثلا:

- allGrades[0][1] مقدارش میشه 18.0 (ردیف 0، ستون 1)
- allGrades[2][0] مقدارش میشه 6.0
- allGrades[3][2] مقدارش میشه 18.0

در زیر برنامه ای کامل برای محاسبه میانگین نمرات 4 دانش آموز نوشته شده. این برنامه نمرات هر دانش آموزو از ورودی دریافت میکنه

```
package main

import "fmt"

func main() {
    var allGrades [4][3]float32
    var averages [4]float32

    fmt.Println("نمره ریاضی، فارسی و تربیت بدنی آرین چیه؟ (به ترتیب وارد کن)")
    fmt.Scanln(&allGrades[0][0], &allGrades[0][1], &allGrades[0][2])

    fmt.Println("نمره ریاضی، فارسی و تربیت بدنی ندا چیه؟ (به ترتیب وارد کن)")
    fmt.Scanln(&allGrades[1][0], &allGrades[1][1], &allGrades[1][2])

    fmt.Println("نمره ریاضی، فارسی و تربیت بدنی امید چیه؟ (به ترتیب وارد کن)")
    fmt.Scanln(&allGrades[2][0], &allGrades[2][1], &allGrades[2][2])

    fmt.Println("نمره ریاضی، فارسی و تربیت بدنی نیما چیه؟ (به ترتیب وارد کن)")
    fmt.Scanln(&allGrades[3][0], &allGrades[3][1], &allGrades[3][2])

    averages[0] = (allGrades[0][0] + allGrades[0][1] + allGrades[0][2]) / 3
    averages[1] = (allGrades[1][0] + allGrades[1][1] + allGrades[1][2]) / 3
    averages[2] = (allGrades[2][0] + allGrades[2][1] + allGrades[2][2]) / 3
    averages[3] = (allGrades[3][0] + allGrades[3][1] + allGrades[3][2]) / 3

    fmt.Println(
        "میانگین نمرات آرین، ندا، امید و نیما به ترتیب: ",
        averages[0], averages[1], averages[2], averages[3],
    )

}
```

این برنامه چیکار می‌کنه؟

- نمرات 3 درس (ریاضی، فارسی، تربیت‌بدنی) برای 4 نفر رو از کاربر می‌گیره
- میانگین نمره‌های هر نفر رو حساب می‌کنه
- بعد میانگین‌ها رو چاپ می‌کنه

1-تعریف آرایه‌ها:

```
var allGrades [4][3]float32
var averages [4]float32
```

allGrades یه آرایه 4*3 که توش نمرات 4 نفر برای 3 درس ذخیره می‌شه

averages یه آرایه 4تایی که میانگین نمرات هر نفر رو نگه می‌داره

2-گرفتن نمره‌ها از کاربر:

```
fmt.Scanln(&allGrades[0][0], &allGrades[0][1], &allGrades[0][2])
```

و مشابه این برای نفرات بعدی.

یعنی نمره‌های هر نفر به ترتیب وارد می‌شه و داخل آرایه ذخیره می‌شه.

3-حساب کردن میانگین:

```
averages[0] = (allGrades[0][0] + allGrades[0][1] + allGrades[0][2]) / 3
```

برای هر دانش‌آموز، نمره‌های سه درس رو جمع می‌کنه و تقسیم بر 3 می‌کنه.

4- چاپ نتیجه:

```
fmt.Println("میانگین نمرات: ", averages[0], averages[1], averages[2], averages[3])
```

میانگین نمرات رو نشون می‌ده به ترتیب: آرین، ندا، امید، نیما

تفاوت Scan با Scanln در خوندن از ورودی

فرق اصلی بین Scan و Scanln اینه که هر دو برای گرفتن ورودی از کاربر استفاده می‌شن، ولی تفاوت‌شون توی **چگونگی خواندن خط ورودی** هست.

- Scan وقتی ورودی می‌گیره، می‌تونه چند مقدار رو از چند خط بخونه. یعنی اگه کاربر چندتا عدد پشت سر هم یا حتی در چند خط وارد کنه، Scan به خوندن ادامه می‌ده تا همه مقادیر مورد نظر رو بگیره.
- اما Scanln فقط ورودی‌های داخل **یک خط** رو می‌خونه و وقتی کاربر Enter زد، خوندن تموم می‌شه.

مثال با Scan

```
var a, b int
fmt.Print("دو عدد وارد کن: ")
fmt.Scan(&a, &b)
```

ورودی

```
140
635
```

```
140 635
```

در استفاده از Scan هر دو روش بالا در وارد کردن ورودی معتبر هست

مثال با Scanln

```
var a, b int
fmt.Print("دو عدد وارد کن: ")
fmt.Scanln(&a, &b)
```

ورودی

```
140 635
```

در استفاده از Scanln فقط روش بالا در وارد کردن ورودی معتبر هست

بازی صفحه ی شانس

صفحه شانس یه بازی ساده و سرگرم‌کننده است که توش یه جدول 3 در 3 داریم و باید جایزه‌ها رو پیدا کنیم.
تصور کن یه جدول کوچیک 3 در 3 داریم، مثل صفحه شطرنج یا جدول بازی XO
داخل این جدول، بعضی خونه‌ها جایزه گذاشته شده — یعنی اعداد 1 — و بقیه خالی هستن — یعنی عدد 0.

هدف بازی اینه که تو باید حدس بزنی جایزه کدوم خونه قرار داره.
برای این کار، شماره ردیف و ستون خونه‌ای که فکر می‌کنی جایزه داره رو وارد می‌کنی.

بعد برنامه بررسی می‌کنه که آیا تو درست حدس زدی یا نه:

- اگه تو اون خونه جایزه باشه (عدد ۱ باشه)، برنامه بهت تبریک می‌گه
- اگه جایزه نباشه، برنامه می‌گه متأسفانه اینجا جایزه نیست

این بازی ساده باعث میشه با مفهوم آرایه دوبعدی آشنا بشی و یاد بگیری چطور داده‌ها رو با استفاده از ردیف و ستون مدیریت کنی.

در ضمن با شرط (if) می‌تونیم ورودی‌های اشتباه یا نامعتبر رو هم کنترل کنیم و به کاربر پیام مناسب بدیم.

```
package main


import "fmt"


func main() {
    var board = [3][3]int{
        {0, 1, 0},
        {0, 0, 0},
        {1, 0, 0},
    }

    var row, col int

    fmt.Println("ردیف و ستونی که فکر میکنی جایزه داره وارد کن")

    fmt.Println(":(2 تا 0 بین) ردیف رو وارد کن ")
    fmt.Scan(&row)
    fmt.Println(":(2 تا 0 بین) ستون رو وارد کن")
    fmt.Scan(&col)

    // شرط چک کردن اینکه جایزه هست یا نه
    if row >= 0 && row <= 2 && col >= 0 && col <= 2 {
        if board[row][col] == 1 {
            fmt.Println("تو برنده شدی!")
        } else {
            fmt.Println("متأسفم، اینجا جایزه نیست.")
        }
    } else {
        fmt.Println("ردیف یا ستون اشتباه وارد شده!")
    }
}
```

تمرین 1: محاسبه میانگین درآمد ماهیانه

فرض کن می‌خوای میانگین درآمد ماهیانه کسب و کارت رو حساب کنی.

برنامه ای بنویس که مقدار درآمد هر ماه رو از ورودی دریافت کنه و در نهایت میانگین درآمدو محاسبه و چاپ کنه

این برنامه رو به سه روش مختلف باید بنویسی و توضیح بدی هر روش چه ویژگی و خصوصیاتی داره

روش 1: استفاده از یک متغیر (مثلا جمع درآمد همه ماه‌ها رو به یک متغیر اضافه کنی)

روش 2: استفاده از 12 متغیر جداگانه برای هر ماه

روش 3: استفاده از آرایه به طول 12

تمرین 2: ترانهاده‌ی ماتریس

برنامه ای بنویس که ابتدا مقادیر یک ماتریس 3 در 3 رو از طریق ورودی دریافت کنه و سپس ترانهاده اونو در خروجی چاپ کنه

ماتریس یه جدول از اعداد هست که به صورت سطر و ستون مرتب شده.
مثلا یه ماتریس 3 در 3 یعنی جدولی با 3 سطر و 3 ستون داریم، مثل:

3	2	1
6	5	4
9	8	7

ترانهاده یعنی سطر و ستون‌های ماتریس جاشون عوض بشه.
یعنی عنصر ردیف i و ستون j در ماتریس جدید میره به ردیف j و ستون i.

مثلا ترانهاده ماتریس قبل ماتریس زیر میشه:

3	6	9
2	5	8
1	4	7

تمرین 3: جمع و تفریق ماتریس

برنامه ای بنویس که ابتدا دو ماتریس A و B که هر دو 3 در 3 هستند از ورودی دریافت کنه وحاصل جمع و حاصل تفریق اونارو محاسبه و چاپ کنه

- A + B
- A – B

برای جمع دو ماتریس هم‌اندازه، عددهای هر خانه رو با هم جمع می‌کنیم.

جمع ماتریس چجوریه؟

برای جمع دو ماتریس هم‌اندازه، عددهای هر خانه رو با هم جمع می‌کنیم.
مثلا:

ماتریس: A

1	2	3
4	5	6
7	8	9

ماتریس: B

9	8	7
6	5	4
3	2	1

ماتریس: A + B

10	10	10
10	10	10
10	10	10

تفریق ماتریس چجوریه؟

مشابه جمع، فقط هر خانه ماتریس دوم رو از ماتریس اول کم می‌کنیم.