محدودیت آرایه

تا اینجا یاد گرفتیم که آرایه یه ساختار داده‌ست که برای نگهداری یه مجموعه از داده‌های هم‌نوع استفاده میشه؛ و معمولاً این داده‌ها از نظر معنا هم تا حدی به هم مربوطن.
مثلاً چی؟ نمرات دروس یک دانش‌آموز، اسامی دوستان یه نفر، یا شاخص توده بدنی (BMI) چند نفر مختلف.
آرایه خیلی خوب بهمون کمک کرد که این داده‌ها رو کنار هم نگه داریم و بهشون دسترسی داشته باشیم یا تغییرشون بدیم.

بیایم با یه مثال ساده، مرور کنیم که چطوری می‌تونستیم به اعضای یه آرایه دسترسی داشته باشیم.

فرض کن آرایه‌ی نمرات یه دانش‌آموز رو اینجوری تعریف کردیم:

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

برای اینکه به هر کدوم از نمره‌ها دسترسی پیدا کنیم، فقط کافیه از **اپراتور []** استفاده کنیم.
فقط باید حواسمون باشه که از محدوده‌ی آرایه خارج نشیم!
تو این مثال، چون فقط 3 عنصر داریم، می‌تونیم از اندیس‌های **0 تا 2** استفاده کنیم.

```
fmt.Println("نمره ریاضی:", grades[0])
fmt.Println("نمره فارسی:", grades[1])
fmt.Println("نمره تربیت بدنی:", grades[2])
```

خب، اینا شدن نمرات آرین در سه درس: ریاضی، فارسی و تربیت بدنی.

اما یه سؤال:
به نظرت بهتر نبود به جای اینکه مجبور باشیم عدد 0، 1 یا 2 رو حفظ کنیم،
مستقیماً از اسم درس‌ها استفاده می‌کردیم؟

مثلاً اگه می‌شد اینجوری بنویسیم، خیلی خوندنش راحت‌تر می‌شد:

```
fmt.Println(“نمره ریاضی”, grades[“math”])
fmt.Println(“نمره فارسی:”, grades[“farsi”])
fmt.Println(“نمره تربیت بدنی:”, grades[“physicalEducation”])
```

- خوانایی برنامه بیشتر می‌شد
- لازم نبود بدونیم ریاضی توی کدوم خونه از آرایه‌ست
- حتی اگه ترتیب دروس عوض می‌شد، برنامه‌مون همچنان درست کار می‌کرد

این دقیقاً همون نقطه‌ایه که آرایه‌ها و slice ها **دیگه جواب نمی‌دن**، و وقتشه که با یه ساختار داده‌ی جدید آشنا بشیم که مشکل نام‌گذاری رو برای همیشه حل می‌کنه...
اون ساختار چیزی نیست جز **map**

روش تعریف map

به طور کلی به دو روش میشه map تعریف کرد.

1-تعریف map به روش literal

```
var name map[keyType]valueType = map[keyType]valueType{
    key1: value1,
    key2: value2,
}
```

var

کلمه‌ی کلیدی برای تعریف متغیر

name

نام متغیر برای دسترسی به اون

یعنی قراره با این اسم به map دسترسی داشته باشیم.

هر اسمی میتونی بذاری (مثل grades)

map[keyType]valueType

نوع map رو مشخص می‌کنی: نوع کلید و نوع مقدار

کلید همون نقشی رو داره که ایندکس های عددی در آرایه

=

علامت انتساب

map[keyType]valueType

اینجا دوباره نوع map رو می‌نویسی، ولی این بار همراه با مقداردهی

{...}

مقدارهای اولیه map — کلید: مقدار

key: value

یه جفت کلید و مقدار — مثل دیکشنری

مثال

```
var grades map[string]float32 = map[string]float32{
     “math”: 14.0,
     “farsi”: 18.0,
     “physicalEducation”: 17.0,
}
```

- grades اسم متغیره
- map[string]float32 یعنی کلید های map از نوع string و مقادیر از نوع float32

نکته: از اونجایی که در این حالت تعریف متغیر و مقدار دهی همزمان انجام میشه نیازی به نوشتن نوع متغیر پس از نام متغیر نیست.

بنابراین میشه مثال رو به شکل زیر بازنویسی کرد

```
var grades = map[string]float32{
     “math”: 14.0,
     “farsi”: 18.0,
     “physicalEducation”: 17.0,
}
```

2-تعریف map با استفاده از make

```
var name = make(map[keyType]valueType)
```

var

کلمه‌ی کلیدی برای تعریف متغیر

name

نام متغیر برای دسترسی به اون

یعنی قراره با این اسم به map دسترسی داشته باشیم.

هر اسمی میتونی بذاری (مثل grades)

=

علامت انتساب

make(...)

تابع مخصوص ساختن map و slice در زبان Go

map[keyType]valueType

نوع map رو مشخص می‌کنی: نوع کلید و نوع مقدار

کلید همون نقشی رو داره که ایندکس های عددی در آرایه

مثال

```
var grades map[string]float32 = make(map[string]float32)
grades[“math”] = 14.0
grades[“farsi”] = 18.0
grades[“physicalEducation”] = 17.0
```

توی این روش، اول میایم map رو با استفاده از make تعریف می‌کنیم و بعد دونه‌به‌دونه کلیدها و مقدارهاش رو بهش اضافه می‌کنیم.

این روش یه تفاوت مهم با روش literal داره:

توی روش literal همه‌چی همون اول یک‌جا تعریف و مقداردهی می‌شه. ولی اینجا اول ساختار map رو می‌سازیم و بعد کم‌کم مقادیرش رو وارد می‌کنیم.

واقعیت اینه که توی این روش، نوشتن کد یه کم بیشتر میشه و باید چند خط اضافه بنویسیم. ولی در عوض، کنترل بیشتری داریم. مثلاً اگه قراره داده‌ها از کاربر گرفته بشن یا به‌صورت پویا از یه جای دیگه (مثل دیتابیس یا فایل) بیان، این روش خیلی بهتر جواب می‌ده.

به‌زبون ساده:

- اگه قراره map از اول تا آخر مقدارهاش مشخص باشن -> برو سراغ روش literal.
- اگه قراره بعداً مقادیرش پر بشن یا پویایی تو کاره -> روش make خیلی به کارت میاد.

مقایسه دو روش تعریف map

ویژگی	روش literal	روش make
ساده و جمع و جور بودن	بله – همه‌چیز یک‌جا تعریف می‌شه	نه – باید جداجدا مقدار بدیم
مناسب برای مقداردهی اولیه	عالیه برای مقداردهی سریع	باید دستی اضافه کنیم
مناسب برای داده های پویا	نه خیلی	مناسب اضافه‌کردن مقادیر در طول برنامه
خوانایی برای مقادیر زیاد	سخت می‌شه اگه مقادیر خیلی زیاد بشن	راحت‌تر و مرتب‌تر می‌شه مدیریت کرد

افزودن، ویرایش و حذف در map

افزودن

فرقی نمی‌کنه که map رو با کدوم روش ساخته باشی، برای **افزودن یک عنصر به map** فقط کافیه از عملگر [] استفاده کنی.

به مثال زیر توجه کن

```
var ages = map[string]uint8 {
     “arian”: 13,
     “nada”: 16,
     “omid”: 11,
     “nima”: 13,
}


ages[“narges”] = 14
```

اینجا در ابتدا با استفاده از روش literal یه map ساختیم برای نگهداری سن بچه‌ها، طوری که با اسم هر کدومشون بتونیم راحت به سنشون دسترسی داشته باشیم.

در آخر متوجه شدیم که یادمون رفته سن **نرگس** رو وارد کنیم، برای همین با عملگر [] اومدیم و خیلی راحت مقدارش رو اضافه کردیم.

یه نکته خیلی مهم:

بر خلاف آرایه و slice ، **طول map در زمان ساخت مشخص نیست** و با اضافه‌کردن عنصر جدید، map خودش به‌صورت خودکار بزرگ میشه.

با این وجود، تابع len همچنان برای map قابل استفاده‌ست و می‌تونی تعداد عناصری که در حال حاضر داخل map وجود دارن رو به‌دست بیاری:

```
var ages = map[string]uint8 {
     “arian”: 13,
     “nada”: 16,
     “omid”: 11,
     “nima”: 13,
}

ages[“narges”] = 14
fmt.Println(len(ages))
```

ویرایش

برای **ویرایش مقدار یه عنصر در map** هم دقیقاً از همون روش استفاده می‌کنیم:
فقط کافیه کلید اون عنصر رو بدونیم، و مقدار جدید رو بهش بدیم.

```
var ages = map[string]uint8 {
     “arian”: 13,
     “nada”: 16,
     “omid”: 11,
     “nima”: 13,
}

ages[“narges”] = 14
fmt.Println(“سن امید قبل از اصلاح:”, ages[“omid”])

ages[“omid”] = 10
fmt.Println(“سن امید بعد از اصلاح:”, ages[“omid”])
```

تو این مثال متوجه شدیم که سن امید رو اشتباه وارد کردیم.
امید در واقع 10 سالشه ولی ما 11 وارد کرده بودیم.
با نوشتن مجدد همون کلید ("omid") و مقدار جدید، مقدار قبلی به‌راحتی جایگزین شد.

حالا که بحث ویرایش داغه، بیایم یه بار دیگه نگاهی به همین مثال بندازیم.
یه اشتباه تایپی هم داشتیم !

اسم «ندا» رو به‌اشتباه nada نوشته بودیم، در حالی که باید neda باشه.

بیایم اصلاحش کنیم:

```
var ages = map[string]uint8 {
     “arian”: 13,
     “nada”: 16,
     “omid”: 11,
     “nima”: 13,
}

ages[“narges”] = 14
fmt.Println(“سن امید قبل از اصلاح:”, ages[“omid”])

ages[“omid”] = 10
fmt.Println(“سن امید بعد از اصلاح:”, ages[“omid”])

ages[“neda”] = 16
fmt.Println(ages)
```

خروجی

```
سن امید قبل از اصلاح: 11
سن امید بعد از اصلاح: 10
map[arian:13 nada:16 narges:14 neda:16 nima:13 omid:10]
```

متوجه شدی چی شد؟

الان ما **دوتا کلید** توی map داریم:

- یکی nada که اشتباه تایپی بود
- یکی neda که اصلاحش کردیم

اما چون map بر اساس **کلید دقیق** کار می‌کنه، هر دوتا رو نگه می‌داره.
یعنی فکر نکن وقتی نوشتی neda = 16، اون یکی رو خودش حذف می‌کنه — نه! هنوز هم nada وجود داره.

نکته مهم:

اگه بخوای کلید یه عنصر رو اصلاح کنی، نمی‌تونی مستقیماً اون کلید رو تغییر بدی.

باید اول عنصر قدیمی رو حذف کنی،بعد عنصر جدید رو اضافه کنی.

حذف

برای حذف کردن یه عنصر از map از تابع delete استفاده می‌کنیم.

ساختار کلی:

```
delete(map, key)
```

یعنی:

- پارامتر اول: اسم map
- پارامتر دوم: کلیدی که می‌خوای حذف بشه

حالا میتونیم عنصری که در مثال قبل اضافه بود (یعنی عنصر با کلید nada) رو حذف کنیم

```
var ages = map[string]uint8 {
     “arian”: 13,
     “nada”: 16,
     “omid”: 11,
     “nima”: 13,
}

ages[“narges”] = 14
fmt.Println(“سن امید قبل از اصلاح:”, ages[“omid”])

ages[“omid”] = 10
fmt.Println(“سن امید بعد از اصلاح:”, ages[“omid”])

ages[“neda”] = 16
fmt.Println(“عناصر قبل از حذف عنصر اضافه:”, ages)

delete(ages, “nada”)
fmt.Println(“عناصر پس از حذف عنصر اضافه:”, ages)
```

پیمایش بر روی map

خب، شاید به‌نظر برسه که چون می‌تونیم روی آرایه یا slice با for ساده پیمایش کنیم، پس روی map هم همین کارو می‌تونیم انجام بدیم.
اما واقعیت اینه که **map تو زبان Go ایندکس عددی نداره**.
یعنی کلیدهای map مثل ایندکس‌های آرایه نیستن که با 0, 1, 2 برن جلو.

روش نادرست

به مثال زیر دقت کن:
اینجا یه map داریم که نام چند خودروی محبوب دنیا و میزان **مصرف سوخت** اون‌ها رو نگه می‌داره. هدف اینه که مدل هر خودرو و مصرف سوختش رو چاپ کنیم.

```
cars := map[string]float32{
    "Toyota Corolla": 6.8,
    "Ford Mustang": 12.0,
    "Tesla Model 3": 0.0,      // چون برقیه
    "BMW X5": 10.5,
    "Kia Rio": 6.5,
}

for i := 0; i < len(cars); i++ {
    fmt.Println("میزان مصرف سوخت خودرو", i, ": ", cars[i]) // ❌ این روش رو map کار نمی‌کنه
}
```

این کد حتی کامپایل هم نمیشه!
دلیلش ساده‌ست: کلیدهای map از نوع string هستن، ولی حلقه‌ی for i := 0 ... داره با عدد int کار می‌کنه.

روش درست

برای پیمایش روی map باید از for range استفاده کنیم.
این حلقه به‌صورت خودکار هر بار یه **کلید** و **مقدار** بهت می‌ده.

ساختارش:

```
for key, value := range myMap {
    fmt.Println("کلید:", key, "- مقدار:", value)
}
```

برای پیمایش روی map ، باید از حلقه‌ی for range استفاده کنی، چون این حلقه به‌صورت خودکار بهت **هر کلید و مقدار** رو یکی‌یکی می‌ده.

اگه فقط کلید برات مهمه:

```
for key := range myMap {
    fmt.Println("فقط کلید:", key)
}
```

پس مثال خودروها رو اینجوری می‌نویسیم:

```
cars := map[string]float32{
    "Toyota Corolla": 6.8,
    "Ford Mustang": 12.0,
    "Tesla Model 3": 0.0,     // چون برقیه
    "BMW X5": 10.5,
    "Kia Rio": 6.5,
}

for model, consumption := range cars {
    fmt.Println("میزان مصرف سوخت خودرو", model, ": ", consumption)
}
```

اگه کلید map عددی باشه چی؟

ممکنه بگی خب، اگه کلیدهای map از نوع int باشن، شاید دیگه بشه با for i := 0... پیمایش کرد.

بیایم تست کنیم:

```
grades := map[int]float32{
    0: 14.0,
    1: 18.0,
    2: 17.0,
}

for i := 0; i < len(grades); i++ {
    fmt.Println(grades[i])
}
```

این برنامه اجرا میشه و حتی خروجی هم می‌ده. ولی یه مشکل جدی داره!

تو Go وقتی یه کلید در map وجود نداشته باشه، مقدار پیش‌فرض نوعش برمی‌گرده.
پس اگه کلیدی وجود نداشته باشه، برای float32، مقدار 0.0 چاپ میشه.

مشکلی که پیش میاد:

بیایم یه مقدار جدید به این map اضافه کنیم، ولی با کلید 4:

```
grades := map[int]float32{
    0: 14.0,
    1: 18.0,
    2: 17.0,
}

grades[4] = 14.50

for i := 0; i < len(grades); i++ {
    fmt.Println(grades[i])
}
```

خروجی

```
14
18
17
0
```

چرا؟ چون:

- len(grades) شده 4
- پس حلقه از 0 تا 3 می‌ره
- ولی ما فقط 0, 1, 2, 4 رو داریم
- ایندکس 3 وجود نداره، پس مقدار پیش‌فرض (0.0) چاپ میشه

پس استفاده از for i روی map اشتباهه، چون:

- هیچ تضمینی نیست که کلیدها پشت‌سرهم باشن (0 تا n-1)
- ممکنه کلیدی حذف شده باشه
- ترتیب map مشخص نیست
- خروجی گمراه‌کننده می‌شه (مثل چاپ عدد 0 بدون اینکه واقعا عضو وجود داشته باشه)

روش درست در همه حال for range:

حتی وقتی کلید int هست، بهتره از for range استفاده کنی:

```
grades := map[int]float32{
    0: 14.0,
    1: 18.0,
    2: 17.0,
}

grades[4] = 14.50

for _, grade := range grades {
    fmt.Println(grade)
}
```

خروجی

```
14
18
17
14.5
```

ترتیب ممکنه متفاوت باشه، ولی همه‌ی مقادیر درست چاپ می‌شن.

عملگرهای قابل استفاده روی map

حالا که با ساختار map آشنا شدیم، یه سؤال مهم پیش میاد:

چه کارهایی می‌تونیم با map انجام بدیم؟ چه عملگرهایی روی map قابل استفاده‌ست؟

برخلاف انواع ساده‌تری مثل int یا slice، map یه ساختار خاصه و فقط تعداد محدودی از عملگرها روی اون قابل استفاده‌ست.

به طور کلی در کار با map فقط اجازه داریم از 3 عملگر استفاده کنیم:

عملگر [] برای دسترسی به مقدار

عملگر == و **عملگر =!** فقط برای بررسی nil بودن یا نبودن map

یعنی به هیچ عنوان نمی‌تونیم دو map رو مستقیماً با هم مقایسه کنیم!

این نکته خیلی مهمه و باید همیشه یادت بمونه.

عملگر ==

این عملگر فقط برای بررسی اینه که آیا یه map مقداردهی شده یا هنوز nil هست.

```
package main


import "fmt"


func main() {
    var grades map[string]float32 = make(map[string]float32)
    fmt.Println(grades == nil) // true ✅ چون خالیه
}
```

عملگر =!

به همین صورت می‌تونیم بررسی کنیم که map مقدار داره یا نه.

```
package main


import "fmt"


func main() {
    var grades map[string]float32 = map[string]float32 {
        “math”: 14.0,
        ”farsi”: 18.0,
        ”physicalEducation”: 17.0,
    }
    fmt.Println(grades != nil) // true ✅ چون خالی نیست
}
```

نکته مهم: تعریف map با var کافی نیست!

اگه فقط با var یه map تعریف کنی ولی مقداردهی اولیه انجام ندی، اون map مقدارش nil خواهد بود و نمی‌تونی مستقیماً بهش مقدار بدی. این کار باعث میشه برنامه موقع اجرا **کرش (panic)** کنه!

```
var grades map[string]float32
grades["math"] = 14.0   // ❌ خطا: assignment to entry in nil map
```

راه درست اینه که قبل از استفاده، map رو با make یا literal مقداردهی کنی:

```
var grades map[string]float32 = make(map[string]float32)
grades["math"] = 14.0   // ✅ بدون مشکل
```

بررسی وجود یک کلید

تصمیم گرفتیم به لیست نمرات، نمراتی که دانش‌آموز در امتحان غایب بوده هم اضافه کنیم. آرین تو امتحان تاریخ غیبت کرده و نمره 0 براش درج شده:

```
var grades map[string]float32 = map[string]float32 {
    “math”: 14.0,
    ”farsi”: 18.0,
    ”physicalEducation”: 17.0,
    “history”: 0.0,
}
```

حالا اگه بیایم نمره تاریخ رو چاپ کنیم، مقدار 0 در خروجی می‌بینیم:

```
fmt.Println(grades[“history”])
```

خروجی

```
0
```

اگه بیایم یه نمره‌ای که هیچوقت ثبت نشده رو چاپ کنیم چطور؟

```
fmt.Println(grades[“algebra”])
```

خروجی

```
0
```

بازم 0 تو خروجی چاپ شد!

اینجا یه سوال پیش میاد: چطور بفهمیم که دانش‌آموز تو امتحان جبر غیبت کرده و نمره 0 گرفته، یا اینکه اصلاً نمره جبر براش ثبت نشده؟

تو Go وقتی توی map دنبال یه کلید می‌گردیم که وجود نداره، به‌جاش مقدار پیش‌فرض نوع داده اون کلید (مثلاً 0 برای float32) برمی‌گرده. یعنی بدون هیچ خطایی!

برای همین، باید یه راهی باشه که بفهمیم کلید واقعاً هست یا نه.

خوشبختانه وقتی از map مقدار می‌گیریم، دو تا مقدار برمی‌گرده:

- اولی همون مقدار مربوط به کلیده،
- دومی یه مقدار بولین (true/false) که میگه کلید وجود داره یا نه.

```
value, ok := myMap["key"]
```

ok اگر true بود یعنی کلید وجود داره، اگر false بود یعنی کلید نیست.

پس برای مثال نمراتمون این‌طوری می‌تونیم چک کنیم:

```
historyGrade, isHistoryEntered := grades[“history”]

if isHistoryEntered {
    fmt.Println(“نمره درس تاریخ وارد شده است”)
    if historyGrade == 0.0 {
        fmt.Println(“دانش آموز در امتحان تاریخ غیبت کرده است”)
    } else {
        fmt.Println(“ نمره دانش آموز در درس تاریخ:”, historyGrade)
    }
} else {
    fmt.Println(“نمره درس تاریخ هنوز وارد نشده است”)
}

algebraGrade, isAlgebraEntered := grades[“algebra”]

if isAlgebraEntered {
    fmt.Println(“نمره درس جبر وارد شده است”)
    if algebraGrade == 0.0 {
        fmt.Println(“دانش آموز در امتحان جبر غیبت کرده است”)
    } else {
        fmt.Println(“ نمره دانش آموز در درس جبر:”, algebraGrade)
    }
} else {
    fmt.Println(“نمره درس جبر هنوز وارد نشده است”)
}
```

مقایسه map با slice

خب حالا که با map آشنا شدیم، یه سوال مهم پیش میاد:
فرق map با slice چیه و کِی باید از کدوم استفاده کنیم؟

بیاید خیلی ساده مقایسه‌شون کنیم:

1- روش دسترسی به داده‌ها

- توی **slice** فقط می‌تونی با عدد (اندیس) به داده‌ها دسترسی داشته باشی.
مثلاً عنصر اول این شکلی خونده می‌شه:

```
grades[0]
```

ولی توی **map** می‌تونی با یه کلید دلخواه (مثلاً string یا عدد) به مقدار موردنظر دسترسی پیدا کنی:

```
grades["math"]
```

2- ترتیب عناصر

- **slice** ترتیب داره. یعنی اولین چیزی که اضافه می‌کنی، توی اندیس 0 قرار می‌گیره و همین‌طور ادامه پیدا می‌کنه. ترتیبش همیشه حفظ می‌شه.
- اما **map** ترتیب خاصی نداره .Go تضمین نمی‌کنه که عناصر map همون‌طوری که اضافه شدن، پشت سر هم باقی بمونن. پس اگه ترتیب برات مهمه، map گزینه‌ی مناسبی نیست.

3- نوع کلید

- توی **slice**اندیس‌ها فقط عدد هستن (0، 1، 2 و ...).
- ولی توی **map** می‌تونی کلید رو خودت انتخاب کنی. مثلاً اسم درس، کد محصول، شماره ملی، یا هر چیز دیگه.

جدول مقایسه

ویژگی	slice	map
نوع کلید/اندیس	عددی (0، 1، 2 و ...)	دلخواه (string، int و ...)
ترتیب عناصر	داره (بر اساس ترتیب ورود)	نداره (ترتیب مشخص نیست)
مناسب برای	لیست‌های مرتب‌شده	جستجوی سریع با کلید خاص
نحوه‌ی دسترسی	با اندیس عددی(mySlice[0])	با کلید (myMap["key"])

خلاصه و جمع‌بندی

اگه با یه لیست ساده به ترتیب سر و کار داری، slice انتخاب خوبیه.
اگه می‌خوای سریع یه مقدار رو با یه کلید خاص پیدا کنی، map خیلی بهتره.

یادت باشه هر کدوم از این ساختارها برای یه کار خاص ساخته شدن، و اینکه بدونی کِی از کدوم استفاده کنی، خیلی بهت کمک می‌کنه که کدات تمیزتر، سریع‌تر و حرفه‌ای‌تر بشه.

تمرین 1: محاسبه میانگین نمرات ندا و امید

ندا و امید تو پایه های تحصیلی مختلف هستن و درساشون با هم فرق میکنه. درس هر کدوم از اون ها تو جدول های زیر لیست شده

نمرات دروس ندا

درس	نمره
فارسی	19.50
دین و زندگی	11.00
زبان خارجی	20.00
تربیت بدنی	16.50
جبر	17.50
ریاضیات گسسته	18.00
حساب دیفرانسیل	16.25
فیزیک	18.50
هندسه	19.00

نمرات دروس امید

درس	نمره
فارسی	20.00
ریاضی	6.00
علوم تجربی	17.75
مطالعات اجتماعی	20.00
دین و زندگی / قرآن	14.00
هنر	19.50
تربیت بدنی	19.00

برنامه ای بنویس که نمرات دروس ندا و امید رو از ورودی به شکل زیر دریافت کنه

```
farsi 19.50
algebra 17.50
```

یعنی اسم هر درس و نمره آن هر دو از ورودی به ازای هر درس دریافت شود.

در ابتدا دروس ندا دریافت می شود. پس از وارد شدن هر نمره از کاربر برنامه سوال می‌شود که آیا تمام شد؟ در صورتی که جواب بله باشد نمرات امید دریافت می‌شود و به همین ترتیب بعد از وارد کردن هر یک از نمرات امید همین سوال مجدد پرسیده می‌شود و در صورتی که جواب بله باشد برنامه وارد فاز محاسبات می‌شود.

بعد از وارد شدن تمام نمرات:

1. میانگین نمرات هر نفر رو جداگانه محاسبه و چاپ کن.
2. حالا بررسی کن کدوم درس‌ها بین ندا و امید مشترکه.
3. فقط همون درس‌های مشترک رو در نظر بگیر، از هر نفر نمره اون درس رو بردار و **میانگین نهایی برای مقایسه** رو حساب کن.
4. در نهایت مشخص کن که کدوم‌یک عملکرد بهتری داشته.

تمرین 2: شمارش حروف الفبا

برنامه‌ای بنویس که یه رشته (string) از کاربر بگیره و تعداد تکرار هر **حرف** رو با استفاده از map بشماره.

مثلاً برای ورودی "hello"خروجی برنامه این باشه:

```
h: 1
e: 1
l: 2
o: 1
```

تمرین 3: تحلیل سنی دوستان

برنامه‌ای بنویس که:

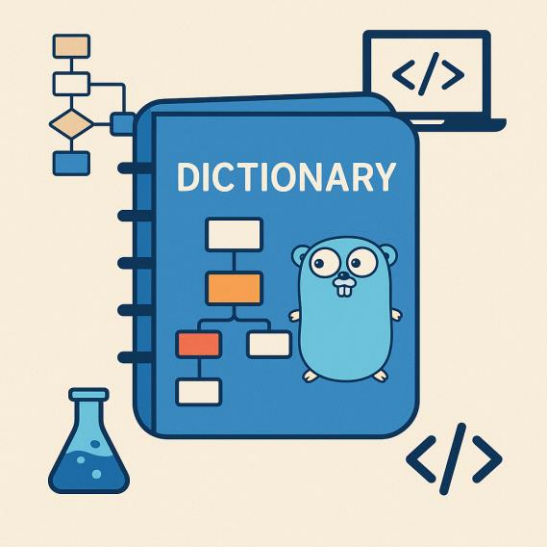
1. اسم و سن دوست‌هات رو در یک map ذخیره کنه.
2. بعد از ثبت اطلاعات، سن بیشترین و کمترین دوست رو چاپ کنه.
   مثلا بگه: بزرگترین دوستت 21 سالشه و کوچیکترینش 16 سالشه.

تمرین 4: حذف دوستان مسن‌تر از میانگین

برنامه تمرین قبل رو گسترش بده:

1. میانگین سنی همه دوست‌ها رو حساب کن.
2. هر دوستی که سنش از میانگین بیشتره رو از map حذف کن.
3. در نهایت جوان‌ترین رو چاپ کن.

تمرین 5: ساخت دیکشنری تعاملی (دو مرحله ای)



یه برنامه دیکشنری بساز که تو **دو مرحله** کار می‌کنه:

مرحله 1 – ساخت دیکشنری

- کاربر باید کلمه‌های انگلیسی و معادل فارسی‌شون رو وارد کنه
- این روند ادامه داره تا وقتی که کاربر --end-- وارد کنه

```
cat
گربه
apple
سیب
--end--
```

مرحله ۲ – ترجمه و افزودن کلمات جدید

- حالا کاربر هر کلمه‌ی انگلیسی که وارد کنه،
  اگه تو دیکشنری باشه، معادل فارسیش رو نشون بده
  اگه نباشه، این پیام رو نشون بده:

```
(yes/no) این کلمه تو دیکشنری نیست. آیا می‌خوای معادل فارسیشو اضافه کنی؟
```

اگه جواب yes بود، معادل فارسی گرفته بشه و به map اضافه بشه.