تا اینجا کلی چیز یاد گرفتیم: متغیر، شرط، حلقه، آرایه، map و...

الان وقتشه که با یه ابزار جدید آشنا بشیم؛ چیزی که قراره قدرت برنامه‌هامون رو چند برابر کنه : **کتابخونه‌ها!**

ولی... اصلاً کتابخونه یعنی چی؟

تو برنامه‌نویسی، کتابخونه (یا همون package) یعنی یه مجموعه از کدهای آماده که یه کار خاص رو برامون انجام می‌دن.

مثلاً یه کتابخونه برای محاسبات ریاضی، یه کتابخونه برای کار با زمان و تاریخ، یا حتی یه کتابخونه برای ساخت رابط گرافیکی.

خبر خوب اینه که زبان Go کلی از این کتابخونه‌های کاربردی رو از قبل آماده کرده و به‌صورت پیش‌فرض همراه خودش داره؛ به اینا می‌گیم **کتابخونه‌های استاندارد**.

یکی از معروف‌ترین و مهم‌ترین این کتابخونه‌ها math هست. همون‌طور که از اسمش پیداست، این کتابخونه مخصوص کارهای ریاضیه:

- می‌خوای جذر بگیری؟ math.Sqrt
- می‌خوای عدد رو گرد کنی؟ math.Round
- می‌خوای عدد π یا e داشته باشی؟ خودش داره!
- حتی برای لگاریتم، سینوس، توان، بزرگ‌ترین عدد و کلی چیز دیگه، توابع آماده داره.

فقط کافیه بگی:

```
import "math"
```

آشنایی با کتابخونه math

تا اینجا فهمیدیم کتابخونه math چه کاربردی داره و چطور باید اون رو وارد کنیم:

تو این بخش پرکاربردترین توابع کتابخونه math در دسته بندی های مختلف لیست شده

دسته‌بندی توابع کاربردی

توابع پایه‌ی ریاضی

این توابع برای محاسبات ساده‌ی عددی هستن، مثلاً جذر، توان، گرد کردن و...

تابع	ورودی	خروجی	توضیح	مثال خروجی
math.Abs(x)	float64	float64	قدر مطلق	math.Abs(-5.5) → 5.5
math.Sqrt(x)	float64	float64	جذر یا ریشه دوم	math.Sqrt(16) → 4
math.Pow(x, y)	float64, float64	float64	x به توان y	math.Pow(2, 3) → 8
math.Ceil(x)	float64	float64	گرد کردن به بالا	math.Ceil(2.3) → 3
math.Floor(x)	float64	float64	گرد کردن به پایین	math.Floor(2.8) → 2
math.Round(x)	float64	float64	گرد کردن به نزدیک‌ترین عدد صحیح	math.Round(2.4) → 2

مقایسه بین عددها

اگر بخوای بین دوتا عدد بیشترین یا کمترین رو پیدا کنی:

تابع	ورودی	خروجی	توضیح	مثال خروجی
math.Max(x,y)	float64, float64	float64	بزرگ‌ترین مقدار بین دو عدد	math.Max(3.5, 9.2) → 9.2
math.Min(x,y)	float64, float64	float64	کوچک‌ترین مقدار بین دو عدد	math.Min(7, 4) → 4

توابع مثلثاتی (Trigonometric)

اگه وارد کارهای پیشرفته‌تر بشی مثل رسم نمودار، کار با بردارها اینا به کارت میان:

تابع	ورودی	خروجی	توضیح	مثال خروجی
math.Sin(x)	float64	float64	سینوس x	math.Sin(math.Pi/2) → 1
math.Cos(x)	float64	float64	کسینوس	math.Cos(0) → 1
math.Tan(x)	float64	float64	تانژانت	math.Tan(0) → 0
math.Asin(x)	float64	float64	آرک‌سینوس	math.Asin(1) → math.Pi/2
math.Acos(x)	float64	float64	آرک‌کسینوس	math.Acos(1) → 0
math.Atan(x)	float64	float64	آرک‌تانژانت	math.Atan(1) → math.Pi/4
math.Hypot(x,y)	float64, float64	float64	فاصله دو نقطه (فیثاغورث)	math.Hypot(3,4) → 5

توابع لگاریتمی و نمایی

برای کارهای آماری، مالی یا علمی ممکنه به این توابع نیاز داشته باشی:

تابع	ورودی	خروجی	توضیح	مثال خروجی
math.Log(x)	float64	float64	لگاریتم طبیعی (پایه e)	math.Log(1) → 0
math.Log10(x)	float64	float64	لگاریتم پایه ۱۰	math.Log10(100) → 2
math.Exp(x)	float64	float64	e به توان x	math.Exp(1) → 2.718281828459045

ثابت‌های آماده

کتابخونه math فقط تابع نداره، کلی مقدار ثابت هم داره که تو ریاضیات کاربرد دارن:

ثابت	مقدار تقریبی	توضیح
math.Pi	3.14159...	عدد π
math.E	2.71828...	عدد e
math.Phi	1.61803...	نسبت طلایی
math.Sqrt2	1.41421...	جذر عدد 2
math.Ln2	0.69314...	لگاریتم طبیعی 2
math.Ln10	2.30258...	لگاریتم طبیعی 10

تولید عدد تصادفی

در زبان Go تولید اعداد تصادفی با استفاده از کتابخونه‌ی زیر انجام می‌شه:

```
import "math/rand"
```

تفاوت math و math/rand چیه؟

در زبان Go کتابخونه‌ها (یا همون پکیج‌ها) معمولاً به صورت جداگانه و منظم توی مسیرهای خاصی دسته‌بندی شدن. مثلاً:

- math -> توابع پایه‌ی ریاضی (مثل Sqrt, Pow, Abs)
- math/rand -> توابع تولید عدد تصادفی

این یعنی پکیج rand زیرمجموعه‌ای از math هست. دقیقاً مثل پوشه‌ای داخل یه پوشه بزرگ‌تر. ولی اینا **دو پکیج کاملاً جدا** هستن.

پس وقتی می‌نویسیم import "math/rand" یعنی چی؟

در Go وقتی می‌نویسیم:

```
import "math/rand"
```

داریم به کامپایلر می‌گیم که:

لطفاً پکیجی به اسم rand که در مسیر math/rand هست رو برای من بیار!

این مسیر (math/rand) مثل آدرس یه پوشه‌ست.

```
import "math/rand"
n := rand.Intn(10)
```

یعنی مهم نیست مسیر math/rand باشه، چیزی که شما تو کد استفاده می‌کنی اسم خود پکیجه rand

تولید عدد تصادفی صحیح

تابع rand.Intn(n) یک عدد **صحیح (int)** بین 0 تا n-1 تولید می‌کنه:

```
import "math/rand"


n := rand.Intn(10)
fmt.Println(n)
```

توابع دیگه در math/rand

تابع	توضیح
rand.Int()	عدد تصادفی از نوع int
rand.Intn(n)	عدد صحیح بین 0 تا n-1
rand.Float32()	عدد اعشاری بین 0.0 تا 1.0
rand.Float64()	عدد اعشاری بین 0.0 تا 1.0
rand.Perm(n)	slice شامل یک ترتیب تصادفی از اعداد 0 تا n-1

جمع‌بندی

کتابخونه‌ی math توی Go مثل یه جعبه‌ابزار کامل برای محاسبات عددیه.
از توابع ساده مثل جذر، قدر مطلق و گرد کردن گرفته تا عملیات پیچیده‌تری مثل لگاریتم، توابع مثلثاتی و توان.

در این درس یاد گرفتیم:

- چطور کتابخونه math رو وارد کنیم و باهاش کار کنیم
- با مهم‌ترین توابع ریاضی، مثلثاتی و لگاریتمی آشنا شدیم
- ثابت‌هایی مثل π و e رو شناختیم
- و یاد گرفتیم که بیشتر این توابع با نوع float64 کار می‌کنن

تمرین 1: جذر نمرات

لیستی از نمرات یک دانش‌آموز در قالب slice داریم.

```
grades := []float64{14.0, 18.0, 17.0}
```

برنامه‌ای بنویس که:

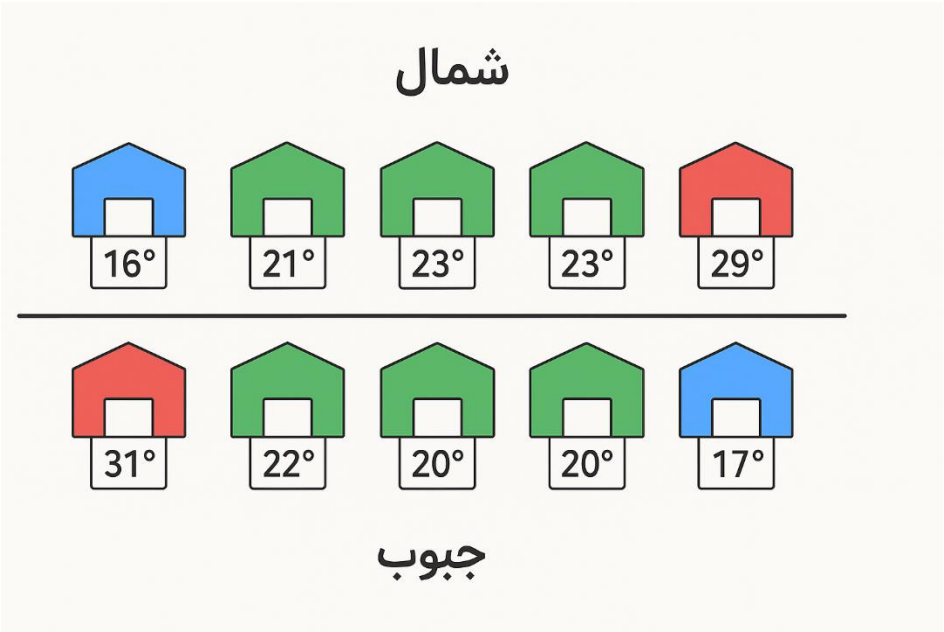
- جذر تمام نمرات رو با استفاده از math.Sqrt محاسبه کنه
- نمرات و جذرشون رو به‌صورت مرتب چاپ کنه

تمرین 2: پیدا کردن بیشترین نمره بین دانش‌آموزها

```
averages := map[string]float64{
    “arian”: 16.32,
    “omid”: 18.0,
    “neda”: 15.0,
    “nima”: 17.9,
}
```

یه map[string]float64 داریم که اسم دانش‌آموزها رو به معدلشون وصل می‌کنه.
برنامه‌ای بنویس که با استفاده از math.Max، بیشترین نمره رو پیدا کنه و اسم کسی که اون نمره رو داره چاپ کنه.

تمرین 3: بررسی دمای سوله های کارخانه



دمای سوله های مختلف یک کارخانه به واحد سلسیوس در یک map ذخیره شده. با استفاده از تابع math.Ceil هر دما را رو به بالا رند کن و سپس گرم ترین و سردترین سوله رو در خروجی چاپ کن.

```
temperatures:= map[string]float64{
    “North-A”: 27.80,
    “North-B”: 34.60,
    “North-C”: 22.40,
    “South-A”: 41.60,
    “South-B”: 20.95,
    “South-C”: 25.82,
}
```

تمرین 4: تبدیل زاویه به رادیان و محاسبه سینوس

برنامه‌ای بنویس که یک slice از زوایا به درجه رو دریافت کنه (مثلاً [0, 30, 45, 60, 90]) برای هر زاویه:

- اون رو به رادیان تبدیل کنه
- با تابع math.Sin سیونس اون زاویه رو محاسبه و چاپ کنه

تمرین 5: تولید قوطی نوشابه

یک کارخانه نوشابه‌سازی قصد داره یه خط تولید جدید راه بندازه و برای این کار، در حال طراحی مدل‌های مختلفی از قوطی‌های نوشابه‌ست.

مهندسا 10 مدل مختلف طراحی کردن که در هر مدل، **شعاع (radius)** و **ارتفاع (height)** قوطی‌ها با هم فرق داره. حالا قراره بررسی کنن که کدوم قوطی‌ها برای نوشابه‌های استاندارد **500 میلی‌لیتری** و **1500 میلی‌لیتری** مناسب هستن.

اطلاعات ابعاد این 10 مدل در ساختار داده ای به شکل زیر در برنامه ذخیره شده

```
cans := []map[string]float32{
    {"radius": 3.6, "height": 12.5},
    {"radius": 4.0, "height": 10.0},
    {"radius": 6.2, "height": 12.0},
    {"radius": 6.5, "height": 12.0},
    {"radius": 5.8, "height": 15.0},
    {"radius": 5.9, "height": 14.7},
    {"radius": 3.2, "height": 4.5},
    {"radius": 4.2, "height": 7.2},
    {"radius": 4.8, "height": 9.3},
    {"radius": 3.5, "height": 4.8},
}
```

وظیفه شما چیه؟

1-با استفاده از فرمول حجم استوانه، برای هر قوطی حجمش رو حساب کن:

```
volume = height * π * (radius)²
```

2-بعد باید قوطی‌هایی رو فیلتر کنی:

- اگر حجم قوطی در حدود **500 میلی‌لیتر** باشد (با تلورانس ±150) و **کمتر از 500 میلی‌لیتر نباشد**، آن قوطی به‌عنوان **قوطی استاندارد 500 میلی‌لیتری** پذیرفته می‌شود.
- اگر حجم قوطی در حدود **1500میلی‌لیتر** باشد (با تلورانس ±150) و **کمتر از 1500 میلی‌لیتر نباشد**، آن قوطی به‌عنوان **قوطی استاندارد 1500 میلی‌لیتری** پذیرفته می‌شود.

3-در نهایت، قوطی‌های مناسب رو توی یه map با ساختار زیر بریز:

```
standardSodaCans := map[uint16][]map[string]float32{
    500:  {},
    1500: {},
}
```

هر قوطی مناسب، باید به شکل زیر ذخیره بشه:

```
{
    "radius": 3.5,
    "height": 4.5,
    "volume": 650.0,
}
```

نکات مهم

- از math.Pi و math.Pow برای محاسبه استفاده کن
- نوع داده‌ی float32 برای radius، height و volume کافیه
- یادت نره که حجم‌ها معمولاً با اعشار هستن، پس float ضروریه

نمونه خروجی

فرض کن بعضی از قوطی‌های مناسب این خروجی رو بدن:

```
{
    500: [
        {
            radius: 3.6,
            height: 12.5,
            volume: 508.93802,
        },
    ],
    1500: [
        {
            radius: 6.5,
            height: 12.0,
            volume: 1592.7875,
        },
    ],
}
```