عملگرهای مقایسه ای

تو هر زبون برنامه نویسی، خیلی وقت‌ها لازم داریم دو مقدار رو با هم مقایسه کنیم تا بفهمیم کدوم بزرگ‌تره، کوچیک‌تره یا برابرند. این کار به ما کمک می‌کنه تا بتونیم تصمیم‌های منطقی بگیریم و برنامه‌هامون رو هوشمندتر کنیم.

برای مثال فرض کن یه برنامه می‌نویسی که شاخص توده بدنی (BMI) رو حساب می‌کنه و عدد 25 رو بهت می‌ده. این عدد به تنهایی خیلی معنی‌دار نیست، ولی اگر برنامه به جای فقط نمایش عدد، بگه:
"**وزنت در محدوده نرمال است** "یا "**اضافه وزن داری**"، فهمیدن وضعیت راحت‌تر می‌شه.

در دنیای برنامه‌نویسی، این کار با **عملگرهای مقایسه‌ای (Comparison Operators)** انجام می‌شه. این عملگرها به ما اجازه می‌دن دو مقدار رو مقایسه کنیم و جواب درست یا غلط (boolean) بگیریم.

به این مثال توجه کن

```
package main


import "fmt"


func main() {
    var arianAge, nedaAge, omidAge, nimaAge uint = 13, 16, 10, 13
    fmt.Println("آرین بزرگتر از امید هست؟", arianAge > omidAge)
    fmt.Println("ندا بزرگتر از آرین هست؟", nedaAge > arianAge)
    fmt.Println("امید کوچیکتر از ندا هست؟", omidAge < NedaAge)
    fmt.Println("آرین کوچیکتر از امید هست؟", arianAge < omidAge)
    fmt.Println("نیما همسن یا بزرگتر از امید هست؟", nimaAge >= omidAge)
    fmt.Println("نیما همسن امید هست؟", nimeAge == omidAge)
    fmt.Println("آرین با نیما همسن نیست؟", ariageAge != nimaAge)
}
```

نکته: خیلی مهمه بدونی که ما به زبان Go یاد نمی‌دیم چطور مقایسه کنه؛ خودش از قبل این قابلیت رو داره! ما فقط ازش می‌خوایم دو مقدار رو مقایسه کنه و نتیجه‌ش رو بهمون بده.

حتی جالب‌تر اینه که Go می‌تونه مقادیر رشته‌ای (string) رو هم با هم مقایسه کنه، نه فقط عددی.

به مثال زیر توجه کن

```
package main


import "fmt"


func main() {
    var ariansFather string = “Majid”
    var nedasFather string = “Majid”
    var omidsFather string = “Majid”
    var nimasFather string = “Masoud”


    fmt.Println(“امید با ندا خواهر/برادر هستند؟”, omidsFather == nedasFather)
    fmt.Println(“نیما با امید برادر هستند؟”, nimasFather == omidsFather)
}
```

فرض کن نمره ریاضی چند تا دانش آموزو داریم و می‌خوایم ببینیم کدوم نمره بیشتره یا کمتره.

```
package main


import "fmt"


func main() {
    var arianGrade, nedaGrade, omidGrade, nimaGrade float64 = 14.0, 18.0, 6, 18.0
    fmt.Println("آرین مردود شد؟", arianGrade < 10.0)
    fmt.Println("ندا مردود شد؟", nedaGrade < 10.0)
    fmt.Println("امید مردود شد؟", omidGrade < 10.0)
    fmt.Println("نیما مردود شد؟", nimaGrade < 10.0)


    fmt.Println("آرین نمرش از امید بهتر شد؟", arianGrade < omidGrade)
    fmt.Println("ندا نمرش مثل نیما شد؟", nedaGrade == nimaGrade)
    fmt.Println("امید نمرش بهتر از ندا شد؟", omidGrade > nedaGrade)
}
```

مثالی که زدم شاید در ظاهر درست باشه، ولی یه مشکل بزرگ داره که خیلی‌ها بهش توجه نمی‌کنن. قبلش بهتره با مفهوم **خطای منطقی (Logical Error)** آشنا بشیم.

خطای منطقی

خطای منطقی یعنی برنامه از نظر ظاهری درست اجرا میشه، هیچ اروری نمی‌گیریم، ولی خروجی یا نتیجه‌ای که می‌گیریم اشتباهه. به عبارت ساده‌تر، برنامه «به‌قول خودش» درست کار کرده، ولی اون کاری که ما واقعاً می‌خواستیم رو به درستی انجام نداده.

مثال خیلی ساده

```
var a = 5
var b = 2
fmt.Println("میانگین دو عدد:", a + b / 2)
```

به نظرت این برنامه میانگین رو به درستی محاسبه میکنه؟ نه! چون طبق اولویت عملگرها ابتدا b تقسیم به 2 میشه و سپس با a جمع میشه.بنابراین شاید در ظاهر داره میانگین رو محاسبه میکنه اما نتیجه میانگین دو عدد نیست.

باید اینطوری می‌نوشتیم:

```
fmt.Println("میانگین دو عدد:", (a + b) / 2)
```

به طور خلاصه:
خطای منطقی (Logical Error) یعنی برنامه اجرا می‌شه، هیچ اروری هم نمی‌ده، ولی نتیجه‌ای که نشون می‌ده اشتباهه.

حالا بریم سراغ مثال نمره

ما نمره‌ی دو نفر رو اینطوری مقایسه کردیم:

```
fmt.Println("ندا نمرش مثل نیما شد؟", nedaGrade == nimaGrade)
```

مشکل کجاست؟

وقتی کامپیوتر عدد اعشاری یا همون float رو ذخیره می‌کنه، همیشه دقیقاً همونی که ما نوشتیم رو نگه نمی‌داره. مثلاً به جای 18.0 ممکنه یه عدد خیلی نزدیک بهش ولی کمی متفاوت ذخیره بشه، چیزی مثل:

```
18.000000000000004
```

چرا این اتفاق می‌افته؟

چون اعداد اعشاری توی کامپیوتر به صورت دودویی ذخیره می‌شن و بعضی اعداد قابل نمایش دقیق به این صورت نیستن. در نتیجه، وقتی دو عدد float رو با == مقایسه می‌کنیم، حتی اگر ظاهراً برابر باشن، ممکنه نتیجه false بشه.

راه حل : مقایسه‌ی بازه‌ای (Range Compare)

برای اینکه مشکل مقایسه‌ی دقیق اعداد اعشاری رو حل کنیم، به جای اینکه بگیم:

آیا دقیقاً برابرند؟ (a == b)

باید بپرسیم:

آیا اختلاف بین این دو عدد خیلی خیلی کوچیکه؟
مثلاً کمتر از 0.000001؟

```
package main


import "fmt"
import "math"


func main() {
    epsilon := 0.000001
    isEqual := math.Abs(nedaGrade - nimaGrade) < epsilon
    fmt.Println("ندا نمرش مثل نیما شد؟", isEqual)
}
```

اینطوری اگه نمرات خیلی نزدیک باشن، مساوی فرض می‌کنیم

وقتی از مقایسه‌ی بازه‌ای استفاده می‌کنیم، یعنی اگه اختلاف نمره‌ها خیلی کم باشه، اون‌ها رو مساوی در نظر می‌گیریم. این روش تو دنیای واقعی خیلی کاربرد داره؛ مثلاً وقتی می‌خوایم نمره، قیمت، وزن یا حتی محاسبات علمی رو با دقت منطقی بررسی کنیم.

جمع‌بندی

- مقایسه‌ی مستقیم عدد اعشاری با == ممکنه باعث خطای منطقی بشه.
- دلیلش اینه که کامپیوتر اعداد اعشاری رو با دقت محدودی ذخیره می‌کنه.
- بهترین راه حل، استفاده از **مقایسه‌ی بازه‌ای (range compare)** با کمک math.Abs هست.
- اگه این نکته رو رعایت نکنیم، ممکنه برنامه بدون هیچ اروری، نتیجه‌ی اشتباهی بهمون بده.

تایپ bool

تا قبل از اینکه با عملگرهای مقایسه‌ای آشنا بشیم، معمولاً خروجی دستورات و پردازش‌هامون عدد صحیح (int)، عدد اعشاری (float) یا رشته (string) بود.

اما وقتی وارد مقایسه‌ها شدیم، با یه نوع داده‌ی جدید به اسم bool آشنا شدیم.

تایپ bool به زبان ساده:

فقط دو مقدار داره:

- true (درست)
- false (نادرست)

یعنی وقتی یک پردازش انجام می‌دیم که نتیجه‌اش فقط می‌تونه درست یا غلط باشه، خروجی اون پردازش از نوع bool هست.

مثال

```
package main


import "fmt"


func main() {
    var isOmidOut bool = true
    fmt.Println("امید خارج از خانه است؟", isOmidOut)
}
```

انواع مقایسه‌ای که می‌تونیم انجام بدیم

وقتی می‌خوایم تو برنامه‌نویسی دو تا چیز (مثل متغیر یا مقدار) رو با هم مقایسه کنیم، خیلی مهمه که نوعشون با هم سازگار باشه.

مثلاً:

- نمی‌تونیم بگیم:
  "سلام" >= 4
  چون یکی رشته (string) و یکی عدد (int) هست و این اصلاً منطقی نیست!
- ولی می‌تونیم بگیم:
  12 < 15
  چون هر دو عدد صحیح (int) هستن و مقایسه‌شون ممکنه.

زبان Go خیلی سخت‌گیره!

یعنی اجازه می‌ده فقط انواعی که هم‌جنس یا قابل تبدیل به هم هستن رو با هم مقایسه کنیم.

چه عملگرهایی برای مقایسه داریم؟

- == (مساوی)
- =! (نابرابر)
- > (کوچکتر)
- => (کوچکتر یا مساوی)
- < (بزرگتر)
- =< (بزرگتر یا مساوی)

جدول مقایسه پذیری

نوع داده	با چی مقایسه میشه؟	عملگرهای مجاز	توضیحات بیشتر
int8, int16, int32, int64, int, rune	با خودشون نظیر به نظیر	==, !=, <, >, <=, >=	نوع‌ها باید یکی باشن یا تبدیل بشن
uint8, uint16, uint32, uint64, uint, byte	با خودشون نظیر به نظیر	==, !=, <, >, <=, >=	نوع‌ها باید یکی باشن یا تبدیل بشن
float32, float64	با خودشون نظیر به نظیر	==, !=, <, >, <=, >=	مقایسه‌ی == ممکنه دقیق نباشه
string	فقط با string	==, !=, <, >, <=, >=	ترتیب الفبایی (Lexicographical)
bool	فقط با bool	==, !=	فقط تساوی و نابرابری مجازه
array	اگر هم نوع و هم طول باشند	==, !=	عناصر مقایسه می‌شن
struct	اگر همه فیلدها قابل مقایسه باشن	==, !=	فیلد به فیلد مقایسه میشه
pointer	با اشاره گرهای هم نوع	==, !=	مقایسه‌ی آدرس
chan	با کانال های هم نوع	==, !=	مقایسه‌ی آدرس کانال
interface	اگر concrete type قابل مقایسه باشه	==, !=	اگر یکی nil باشه، بررسی می‌کنه
slice	نمی‌تونن مستقیماً مقایسه بشن	فقط == nil یا != nil	فقط بررسی وجود/عدم وجود
map	نمی‌تونن مستقیماً مقایسه بشن	فقط == nil یا != nil	فقط بررسی وجود/عدم وجود
function	نمی‌تونن مستقیماً مقایسه بشن	فقط == nil یا != nil	فقط بررسی وجود/عدم وجود

مثال

```
package main

import "fmt"

func main() {
    var (
        wallColor string = “white”
        doorColor string = “brown”
        windowColor string = “brown”
    )

    fmt.Println("رنگ دیوار بزرگتر از رنگ پنجرست؟", wallColor > windowColor)
    fmt.Println("رنگ پنجره بزرگتر از رنگ دیواره؟", windowColor > wallColor)
    fmt.Println("رنگ در و پنجره مثل همن؟", doorColor == windowColor)
    fmt.Println("رنگ دیوار و پنجره مثل هم نیستن؟", wallColor != windowColor)

    var (
        isDoorOpen bool = false
        isWindowOpen bool = true
    )

    fmt.Println("در باز هست؟", isDoorOpen == true)
    fmt.Println("در و پنجره هر تو یه وضعیت قرار دارن؟", isDoorOpen == isWidnowOpen)

    isDoorOpen = true
    fmt.Println("در و پنجره هر تو یه وضعیت قرار دارن؟", isDoorOpen == isWidnowOpen)
}
```

نکته مهم درباره مقایسه رشته‌ها

وقتی دو رشته رو با هم مقایسه می‌کنیم، اول از همه حرف اول هر رشته رو با هم مقایسه می‌کنیم. رشته‌ای که حرف اولش **از نظر ترتیب حروف الفبا** جلوتر باشه، کوچیک‌تر محسوب می‌شه.

مثلاً:
کلمه "brown" کوچیک‌تر از "wall" هست چون حرف b جلوتر از w توی الفبا قرار داره.

اگه دو رشته تا جایی که مقایسه می‌کنیم دقیقاً مثل هم باشن یعنی تا جایی که بررسی می‌کنیم همه حروف برابرن، ولی طول رشته‌ها فرق داره؛ تو این حالت رشته‌ای که طولش بزرگ‌تره، بزرگ‌تر حساب می‌شه.

مثلاً:
"news" بزرگ‌تر از "new" هست چون طولش بیشتره، حتی با اینکه حروف اولشون یکیه.

مثال

```
package main

import "fmt"

func main() {
    var (
        wallColor string = “white”
        doorColor string = “brown”
        windowColor string = “brown”
    )

    fmt.Println("رنگ دیوار بزرگتر از رنگ پنجرست؟", wallColor > windowColor)
    fmt.Println("رنگ پنجره بزرگتر از رنگ دیواره؟", windowColor > wallColor)
    fmt.Println("رنگ در و پنجره مثل همن؟", doorColor == windowColor)
    fmt.Println("رنگ دیوار و پنجره مثل هم نیستن؟", wallColor != windowColor)

    var (
        isDoorOpen bool = false
        isWindowOpen bool = true
    )

    fmt.Println("در باز هست؟", isDoorOpen == true)
    fmt.Println("در و پنجره هر تو یه وضعیت قرار دارن؟", isDoorOpen == isWidnowOpen)

    isDoorOpen = true
    fmt.Println("در و پنجره هر تو یه وضعیت قرار دارن؟", isDoorOpen == isWidnowOpen)
}
```

مقایسه انواع غیر همسان

وقتی می‌خوایم دو مقدار رو با هم مقایسه کنیم که نوع‌شون یکی نیست، اول باید هر دو رو به یه نوع مشترک تبدیل کنیم. فقط بعدش می‌تونیم مقایسه کنیم.

مثلاً:

- اگه بخوایم عدد صحیح ۸ بیتی (int8) رو با عدد صحیح ۳۲ بیتی (int32) مقایسه کنیم، اول باید یکی رو تبدیل کنیم به نوع اون یکی.

مثال

```
package main

import "fmt"
import “strconv”
import "math"

func main() {
    var length int = 10
    var width float32 = 8.5
    var height string = “10”
    numericHeight, _ := strconv.Atoi(height)
    fmt.Println("طول مکعب با ارتفاعش برابره؟", length > numericHeight)
    fmt.Println("عرض مکعب بزرگتر از طولش هست؟", width > float32(length))
    // مقایسه زیر ممکنه باعث خطای منطقی بشه
    fmt.Println("عرض مکعب با ارتفاعش برابر نیست؟", width != float32(numericHeight))
    //این مقایسه بهتر از حالت قبله
    epsilon := 0.000001
    isNotEqual := math.Abs(strconv.Ftoa(width) - height) > epsilon
    fmt.Println("عرض مکعب با ارتفاعش برابر نیست؟", isNotEqual)
}
```

چرا باید این تبدیل انجام بشه؟

چون زبان Go خیلی سخت‌گیر هست و اجازه نمی‌ده بدون تبدیل، انواع متفاوت رو با هم مقایسه کنیم. این کار باعث می‌شه خطاهای احتمالی توی برنامه کمتر بشه.

نکته: با وجود اینکه مقدار height یک عدد صحیح هست، اما چون به صورت رشته ذخیره شده، برای مقایسه باید اول اون رو تبدیل کنیم.

تبدیل رشته و عدد با strconv

اگر بخوایم:

1. یک مقدار عددی (int یا float) رو به رشته تبدیل کنیم
2. یا یک رشته‌ای که عدد داخلش هست رو به عدد تبدیل کنیم

باید از توابع کتابخانه strconv استفاده کنیم، مثل:

- strconv.Itoa(int) برای تبدیل عدد صحیح به رشته
- strconv.ParseFloat(string, bitSize) برای تبدیل رشته به عدد اعشاری
- strconv.Atoi(string) برای تبدیل رشته به عدد صحیح

به مثال زیر توجه کنید

```
package main


import "fmt"
import “strconv”


func main() {
    var masoudWeight string = “84”
    var masoudAge int = 37
    |numericWeight, _ := strconv.Atoi(masoudWeight)
    fmt.Println(":وزن مسعود", numericWeight)
    fmt.Println(":سن مسعود", strconv.Itoa(masoudAge))
}
```

ترکیب شرط‌ها

گاهی پیش میاد که بخوایم چند شرط رو با هم ترکیب کنیم و بررسی کنیم.
مثلاً:

- آیا رنگ پنجره قهوه‌ای **و** رنگ درب قهوه‌ای هست؟
- آیا رنگ دیوار سفید **و** رنگ پنجره قهوه‌ای هست؟
- آیا درب باز **یا** پنجره باز هست؟
- آیا درب باز هست **و** پنجره باز نیست؟

چطوری این کار رو انجام بدیم؟

برای ترکیب چند شرط، از **عملگرهای ترکیب مقایسه** استفاده می‌کنیم.

اسمشون هم گویاست: این عملگرها کمک می‌کنن چند مقایسه رو با هم ترکیب کنیم و یه شرط کلی بسازیم.

سه عملگر اصلی برای ترکیب شرط‌ها:

- && یعنی **و** (AND)
- || یعنی **یا** (OR)
- ! یعنی **نقیض** (NOT)

عملگر && (یعنی "و")

این عملگر به دو طرف خودش نگاه می‌کنه، و فقط وقتی نتیجه‌اش true می‌شه که هر دو طرف شرط‌ها true باشن.

```
var windowsColor, doorColor string = “brown”, “brown”
windowColor == “brown” && doorColor == “brown”
```

چرا حاصل این عبارت true می‌شه؟

چون:

- windowColor == "brown" خودش true هست
- و doorColor == "brown" هم true هست

عملگر && (و) فقط وقتی true می‌شه که **هر دو طرفش** درست باشن.

پس چون هر دو شرط درست هستن، کل عبارت نتیجه‌ش می‌شه true

اگر هر کدوم از شرط‌ها غلط بود، نتیجه کل عبارت می‌شد false

عملگر || (یعنی "یا")

این یه عملگر منطقی هست که دو عملوند می‌خواد. کارش اینه که وقتی حداقل یکی از عملوندها true باشه، نتیجه کل عبارت رو true می‌کنه.

مثال

```
var isDoorOpen, isWindowOpen bool = false, true
isDoorOpen == true || isWindowOpen == true
```

چرا حاصل این عبارت true هست؟

چون:

- isDoorOpen == true نادرسته، چون درب بسته است
- ولی isWindowOpen == true درست هست و به تنهایی کافیه که کل عبارت true بشه.

پس وقتی حداقل یکی از شرط‌ها درست باشه، عملگر || خروجی رو true می‌کنه.

عملگر ! (یعنی "نقیض")

این عملگر فقط **یه عملوند** داره و کارش اینه که مقدار عملوند رو برعکس می‌کنه!
یعنی:

- اگر مقدار true باشه، ! باعث می‌شه نتیجه false بشه
- اگر مقدار falseباشه، ! باعث می‌شه نتیجه true بشه

مثال

```
var isDoorOpen, isWindowOpen bool = false, true

isDoorOpen == true || !isWindowOpen == true
```

چرا حاصل این عبارت false هست؟

بیایم ساده‌سازی کنیم:

- isDoorOpen == true
  یعنی false == true که می‌شه false
- isWindowOpen == true
  یعنی true == true که می‌شه true
- پس عبارت می‌شه:
  false || !true
- حالا !true یعنی false
- پس عبارت نهایی:
  false || false
  که می‌شه false

نکته مهم:

وقتی از عملگر ! استفاده می‌کنیم، باید دقت کنیم که جای عملوند رو درست قرار بدیم و اولویت عملگرها رو رعایت کنیم تا نتیجه درست بشه.

ساختار شرطی

تا اینجا متوجه شدیم که چطور مقایسه مقادیر و متغیرهای مختلف رو انجام بدیم و حتی بتونیم این مقایسه هارو ترکیب کنیم. اما این به تنهایی کافی نیست! نیاز هست با توجه به نتیجه مقایسه که انجام دادیم تصمیم بگیریم و یه کاری انجام بدیم. اون مثالی که راجب نتیجه شاخص توده بدنی زدیم یادته؟ نیازه که نتیجه عددی شاخص توده بدنی رو ارزیابی کنیم و بر اساس اون ارزیابی یه عبارت در خروجی نشون بدیم، این دقیقا همون جایی هست که شرط به کمک ما میاد

ساختار شرطی در ساده ترین حالت خودش به شکل زیر هست

```
if condition {
    statement
}
```

- if یکی از **کلمات کلیدی** (keyword) در زبان Go هست و همیشه باید قبل از شرط نوشته بشه. این کلمه به زبان می‌گه: اگر شرطی برقرار بود، یه کاری انجام بده
- condition همون **عبارت شرطی** یا **مقدار بولی** (bool) هست که نتیجه‌اش فقط می‌تونه یکی از دو حالت true یا false باشه. این مقدار معمولاً از یک **عبارت مقایسه‌ای** به دست میاد.
- {} این بخش بهش می‌گیم **بدنه شرط** یا **بلوک کد**، یعنی اون قسمت از برنامه که فقط وقتی شرط برقرار باشه (یعنی true باشه) اجرا می‌شه.
- statement همون **دستور اجرایی** داخل بدنه‌ی شرطه. می‌تونه یه خط باشه یا چند خط، که برنامه فقط وقتی اون‌ها رو اجرا می‌کنه که شرط درست باشه.

فرض کن داریم یه برنامه برای **هوشمندسازی خونه** می‌نویسیم. قراره این برنامه قبل از اینکه صاحب‌خونه از خونه خارج بشه، بررسی کنه:

- آیا درب بازه؟
- آیا پنجره بازه؟

اگه هرکدوم باز بودن، باید **هشدار** بده که "درب یا پنجره رو ببند."

```
var isDoorOpen, isWindowOpen bool = true, false
if isDoorOpen || isWindowOpen {
    fmt.Println("درب یا پنجره باز هستش، قبل از خروج باید اون هارو ببندی")
}

fmt.Println("به سلامت برگردی!")
```

با اجرای این دستور این خروجی رو مشاهده میکنیم.

```
درب یا پنجره باز هستش، قبل از خروج باید اون هارو ببندی
به سلامت برگردی!
```

این ساختار خیلی خوب و عالی داره کار میکنه اما شاید نیاز باشه در صورتی که درب یا پنجره باز نبود برای اطمینان خاطر بهش پیغامی مبنی براینکه "همه چی خوب و عالیه میتونی خارج بشی" نشون بدیم. در این صورت نیاز هست که از ساختار کامل شرطی استفاده کنیم

ساختار کامل شرطی

وقتی می‌خوایم بگیم:

- اگر **یه شرط خاص برقرار بود**، یه کاری انجام بده
- **در غیر این صورت**، یه کار دیگه انجام بده

از if ... else استفاده می‌کنیم.

```
if condition {
    statement
} else {
    statement
}
```

- if یکی از **کلمات کلیدی** (keywords) زبان Go هست که همیشه قبل از عبارت شرطی نوشته می‌شه.
  این بخش مشخص می‌کنه که اگر شرط برقرار بود، کد داخل بدنه اجرا بشه.
- condition همون **عبارت شرطی**، نتیجه‌ی یک **عبارت مقایسه‌ای یا منطقی** هست.
  این مقدار باید از نوع boolباشه و فقط می‌تونه trueیا falseباشه.
- {} مشخص‌کننده‌ی **بدنه‌ی شرط** هست. یعنی مجموعه دستورات (statements) که فقط در صورت درست بودن شرط (true) اجرا می‌شن.
- statement همون **دستوراتی** هستن که داخل بدنه نوشته می‌شن؛ مثلاً نمایش پیام، تغییر مقدار متغیر یا اجرای یک تابع.
- else هم یکی دیگه از **کلمات کلیدی** زبان Go هست. این بخش زمانی اجرا می‌شه که شرط داخل ifبرقرار **نباشه** (یعنی مقدار false داشته باشه)
- بدنه‌ی else هم با {} مشخص می‌شه و داخلش می‌تونیم دستوراتی بنویسیم که فقط در صورت **نادرست بودن شرط** اجرا می‌شن.
- statement های داخل else هم مثل همیشه همون **دستوراتی** هستن که برنامه در صورت نیاز اجراشون می‌کنه.

نکته : بعد از کلمه کلیدی elseدیگه نیازی به نوشتن شرط (condition) نیست. چون این بخش به صورت پیش‌فرض **همون شرط** if رو دنبال می‌کنه، با این تفاوت که:

دستورات داخل else فقط **زمانی اجرا می‌شن که شرط if برقرار نباشه** (یعنی false باشه)

```
var isDoorOpen, isWindowOpen bool = false, false
if isDoorOpen || isWindowOpen {
    fmt.Println("درب یا پنجره باز هستش، قبل از خروج باید اون هارو ببندی")
} else {
    fmt.Println("همه چی خوب و عالیه میتونی خارج بشی")
}

fmt.Println("به سلامت برگردی!")
```

خروجی این برنامه در زیر آورده شده است

```
همه چی خوب و عالیه میتونی خارج بشی
به سلامت برگردی!
```

نکته:

عبارت "به سلامت برگردی!" همیشه در خروجی چاپ می‌شه،
چون **خارج از ساختار شرطی** نوشته شده و به هیچ شرطی وابسته نیست.
فرقی نداره شرط داخل if برقرار باشه یا نه، این خط همیشه اجرا می‌شه.

نکته:

ساختار شرطی (if / else) می‌تونه **ترتیب معمول اجرای دستورات** در برنامه رو کمی تغییر بده.
همون‌طور که توی مثال دیدیم، چون حاصل condition مقدار false بود، **دستورات داخل بدنه‌ی if** **اجرا نشدن** و برنامه به‌جای اون، رفت سراغ بخش else

ساختار شرطی تو در تو (Nested if)

گاهی وقتا لازمه **اول یه شرط کلی رو بررسی کنیم** و **فقط در صورتی که اون برقرار بود**، بریم سراغ بررسی یه شرط دیگه.

در این حالت از **ساختار شرطی تو در تو** استفاده می‌کنیم. یعنی if داخل یه if دیگه

به مثال زیر توجه کنید

```
package main


import "fmt"


func main() {
    var isDoorOpen, isWindowOpen bool = true, false
    var isGoingOut bool = true
    if isGoingOut == true {
        if isDoorOpen || isWindowOpen {
            fmt.Println("درب یا پنجره باز هستش، قبل از خروج باید اون هارو ببندی")
        } else {
            fmt.Println("همه چی خوب و عالیه میتونی خارج بشی")
        }
        fmt.Println("به سلامت برگردی!")
    }
}
```

ساختارهای شرطی تو در تو از مثال قبل پیچیده تر هم میتونن باشن، به عنوان مثال فرض کن بر اساس نمره ای که یک فرد در درس ریاضی گرفته میخوایم بهش تعدادی شکلات جایزه بدیم.

```
package main

import "fmt"

func main() {
    var omidMathGrade float32 = 14.75
    if omidMathGrade >= 18.0 {
        fmt.Println("4 تا شکلات میگیری!")
    } else {
        if omidMathGrade >= 15.0 {
            fmt.Println("3 تا شکلات میگیری!")
        } else {
            if omidMathGrade >= 12.0 {
                fmt.Println("2 تا شکلات میگیری")
            } else {
                if omidMathGrade >= 10.0 {
                    fmt.Println("1 شکلات میگیری")
                } else {
                    fmt.Println("باید بیشتر تلاش میکردی!")
                }
            }
        }
    }
}
```

درسته که ساختار قبلی ما درست کار می‌کرد، ولی وقتی چند تا شرط پشت سر هم داریم، شکل تو در تو باعث می‌شه کد شلوغ و سخت‌خوان بشه. زبان Go برای این جور شرایط یه راهکار ساده داره یعنی استفاده از ساختار else if.

```
package main

import "fmt"

func main() {
    var omidMathGrade float32 = 14.75

    if omidMathGrade >= 18.0 {
        fmt.Println("تا شکلات میگیری 4!")
    } else if omidMathGrade >= 15.0 {
        fmt.Println("تا شکلات میگیری 3!")
    } else if omidMathGrade >= 12.0 {
        fmt.Println("تا شکلات میگیری 2")
    } else if omidMathGrade >= 10.0 {
        fmt.Println("شکلات میگیری 1")
    } else {
        fmt.Println("باید بیشتر تلاش میکردی!")
    }
}
```

مفهوم Variable Scope داخل ساختار شرطی

وقتی از ساختار شرطی استفاده می‌کنیم، ممکنه هدفمون فقط **تعیین مقدار یه متغیر** باشه، نه لزوماً اجرای چند خط کد یا چاپ مستقیم.

چرا این کار مهمه؟

- با این روش **کد کمتر تکراری می‌شه**
- همون مقدار متغیر رو می‌تونیم بعداً **یک بار چاپ کنیم** یا هر جای دیگه استفاده کنیم
- باعث می‌شه کد مرتب‌تر و قابل نگهداری‌تر باشه

به مثال زیر توجه کن

```
package main

import "fmt"

func main() {
    var province string = “fars”

    if province == “tehran” {
        var city string = “tehran”
    else if province == “fars” {
        var city string = “shiraz”
    else if province == “isfahan” {
        var city string = “isfahan”
    else if province == “guilan” {
        var city string = “rasht”
    } else {
        var city string = “unknown”
    }

    fmt.Println(“مرکز استان ”, province, “ شهر ”, city, “است ”)
}
```

کد نوشته شده در این مثال، قابل اجرا (کامپایل) نیست. متغیر city داخل هر بلوک if یا else if به صورت جداگانه تعریف شده و این به این معنی هست که خارج از اون بلاک این متغیر انگار که هیچوقت تعریف نشده. دستور چاپ داره به متغیر city ارجاع میکنه که هیچوقت تعریف نشده

به زبان ساده تر:

- هر var city داخل یک بلوک {} تعریف شده،
- **آن متغیر فقط در همان بلوک قابل دسترس است** (scoop محلی دارد)
- وقتی بیرون بلوک می‌خوای به city دسترسی داشته باشی، کامپایلر می‌گه چنین متغیری وجود نداره!

راه حل درست

برای اینکه بتونیم بعد از شرط مقدار city رو استفاده کنیم، باید اول **یک بار خارج از شرط متغیر رو تعریف کنیم**

سپس داخل هر شرط فقط بهش مقدار بدیم

```
package main

import "fmt"

func main() {
    var province string = “fars”
    var city string

    if province == “tehran” {
        city = “tehran”
    else if province == “fars” {
        city = “shiraz”
    else if province == “isfahan” {
        city = “isfahan”
    else if province == “guilan” {
        city = “rasht”
    } else {
        city = “unknown”
    }

    fmt.Println(“مرکز استان ”, province, “ شهر ”, city, “است ”)
}
```

ساختار switch case

وقتی چند شرط مختلف داریم که همگی قراره فقط یک مقدار خاص (مثلاً مقدار متغیری مثل province) رو بررسی کنن استفاده از switch انتخابی بهتر و مرتب‌تر نسبت به چندین if پشت سر هم هست

چرا از switch استفاده کنیم؟

- کد کوتاه‌تر و خواناتر می‌شه
- از تکرار کلمه ifجلوگیری می‌کنه
- سرعت اجرا هم معمولاً بهتره چون فقط یه بار مقدار چک می‌شه و سریع‌تر تصمیم گرفته می‌شه
- نگهداری و توسعه برنامه راحت‌تر می‌شه

```
switch comparisonValue {
    case value1:
        statement
    case value2:
        statement
    default:
        statement
}
```

switch یک **دستور کنترلی (Control Statement)** است که برای **انتخاب یکی از چند مسیر اجرایی** بر اساس مقدار یک عبارت استفاده می‌شود.

comparisonValue همان **عبارت مورد مقایسه (expression to evaluate)** است که مقدار آن با مقادیر داخل هر case مقایسه می‌شود.

هر case یک **شاخه (branch)** از اجرای برنامه است که زمانی فعال می‌شود که مقدار comparisonValue با مقدار case برابر باشد.

statement داخل هر case همان **دستورات اجرایی (executable statements)** مربوط به آن شاخه است.

بخش default یک **شاخه پیش‌فرض (default branch)** است که وقتی هیچ‌کدام از شرایط case برقرار نباشد، اجرا می‌شود.

این بخش مشابه آخرین else در ساختار if else if else if ... else می باشد.

کی از switch case استفاده کنیم؟

- وقتی **می‌خوایم یک مقدار رو با چند مقدار مشخص مقایسه کنیم**
- وقتی شرط‌ها **روی یک متغیر یا عبارت واحد هستند** و می‌خوایم به جای چندتا if پشت سر هم، کد مرتب‌تر و خواناتری داشته باشیم
- وقتی تعداد حالت‌ها (شرط‌ها) **زیاده و استفاده از if طولانی و پیچیده می‌شه**
- وقتی می‌خوایم کد سریع‌تر اجرا بشه چون switch در بیشتر زبان‌ها (از جمله Go) بهینه‌تر از چندین if متوالی هست
- وقتی می‌خوایم کد تمیزتر، قابل فهم‌تر و آسان‌تر برای نگهداری بنویسیم

کی از switch case استفاده نکنیم؟

- وقتی شرایط مقایسه‌ای پیچیده داریم (مثلاً ترکیب چند شرط با && یا ||)
- وقتی هر شرط روی متغیرهای متفاوت یا انواع مختلف داده‌ست
- وقتی شرط‌ها شامل بازه‌های عددی یا مقایسه‌های غیر مساوی هستن

مثال قبل رو با استفاده از ساختار switch case در زیر بازنویسی کردیم

```
package main

import "fmt"

func main() {
    var province string = “fars”
    var city string

    switch province {
        case “tehran”:
            city = “tehran”
        case “fars”:
            city = “shiraz”
        case “isfahan”:
            city = “isfahan”
        case “guilan”:
            city = “rasht”
        default:
            city = “unknown”
    }

    fmt.Println(“مرکز استان ”, province, “ شهر ”, city, “است ”)
}
```

فرض کن میخوایم برنامه را به جوری تغییر بدیم که تو حالت هایی که حرف اول اسم هر استان حرف بزرگ بود بازم برنامه به درستی کار کنه. میتونیم برنامه رو به شکل زیر بازنویسی کنیم.

```
package main

import "fmt"

func main() {
    var province string = “fars”
    var city string

    switch province {
        case “tehran”:
            city = “tehran”
        case “Tehran”:
            city = “tehran”
        case “fars”:
            city = “shiraz”
        case “Fars”:
            city = “shiraz”
        case “isfahan”:
            city = “isfahan”
        case “Isfahan”:
            city = “isfahan”
        case “guialn”:
            city = “rasht”
        case “Guilan”:
            city = “rasht”
        default:
            city = “unknown”
    }

    fmt.Println(“مرکز استان ”, province, “ شهر ”, city, “است ”)
}
```

ادغام چند case در ساختار switch

گاهی پیش میاد که چند مقدار مختلف باید یک کار مشترک انجام بدن.
مثلاً توی مثال استان‌ها، دو مقدار "fars" و "Fars" هر دو باید باعث بشن "shiraz" = city بشه.

ساختار switch case بهمون این قابلیت رو داده که بتونیم چند شرط رو باهم ترکیب کنیم و از تکرار case ها جلوگیری کنیم. مثال قبل رو متونیم با همین ویژگی بازنویسی کنیم تا از تکرار case ها جلوگیری کنیم. تو این حالت کدساده تر و خواناتر شده و همچنین تعداد case ها از 8 به 4 کاهش پیدا کرد.

```
package main


import "fmt"


func main() {
    var province string = “Fars”
    var city string


    switch province {
        case “tehran”, “Tehran”:
            city = “tehran”
        case “fars”, “Fars”:
            city = “shiraz”
        case “isfahan”, “Isfahan”:
            city = “isfahan”
        case “guialn”, “Guilan”:
            city = “rasht”
        default:
            city = “unknown”
    }


    fmt.Println(“مرکز استان ”, province, “ شهر ”, city, “است ”)
}
```

تمرین 1: بررسی خطر آدم ربایی

فرض کن اطلاعات چهار نفر داریم: آرین، ندا، امید و نیما.
می‌خوایم برنامه‌ای بنویسیم که **خطر آدم‌ربایی** برای هر کدوم رو حساب کنه.
اگر کسی در خطر بود، مثلاً بنویسه:

آرین در خطر آدم ربایی هست

نام	سن	شغل پدر	مناطق مجاز برای حضور
آرین	13	بازرگان	مرکز خرید
ندا	16	بازرگان	کلاس شنا
امید	10	بازرگان	شهربازی
نیما	13	دامدار	کلاس تنیس

شرایط خطر آدم ربایی چیه؟

یه نفر وقتی **در خطر آدم‌ربایی هست** که همزمان:

- **موقعیت مکانی**ش توی شهربازی یا مرکز خرید باشه
- **شغل پدر**ش بازرگان باشه
- و **سن**ش کمتر از 15 سال باشه

برای اینکه شخص در خطر باشه، باید **همه‌ی ۳ شرط** بالا با هم برقرار باشن.

تمرین 2: میانگین نمرات

جدول نمرات چند درس برای آرین، ندا، امید و نیما:

	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

هدف

میانگین نمرات هر فرد رو حساب کنیم، بعد بر اساس میانگینشون یه ارزیابی چاپ کنیم:

بر اساس میانگین هر یک از آن ها، نتیجه زیر چاپ شود

شرط	نتیجه
میانگین بزرگتر/مساوی 18.0 باشد	عالی
میانگین بزرگتر/مساوی 16.0 و کوچکتر از 18.0 باشد	خوب
میانگین بزرگتر/مساوی 13.0 و کوچکتر از 16.0 باشد	بد
میانگین بزرگتر/مساوی 10.0 و کوچکتر از 13.0 باشد	نیاز به تلاش بیشتر
میانگین کمتر از 10.0 باشد	افتضاح

تمرین 3: محاسبه شاخص توده بدنی (BMI)

برنامه‌ای بنویس که قد (به متر) و وزن (به کیلوگرم) کاربر رو دریافت کنه، شاخص توده بدنی (BMI)رو محاسبه کنه.

فرمول محاسبه BMI:

```
bmi = weight / height * height
```

نکته: متغیر قد و وزن هر دو از نوع اعشاری است.

با توجه به مقدار محاسبه شده و جدول زیر خروجی معنادار چاپ شود

شرط	نتیجه
BMI بزرگتر از 25.0 باشه	اضافه وزن داری
BMI بزرگتر/مساوی 18.5 و کوچکتر/مساوی 25 باشه	وزنت مناسبه
BMI کوچکیتر از 18.5 باشه	باید وزنتو بیشتر کنی

تمرین 4: ترجمه رنگ

طبق جدول زیر برنامه ای بنویسید که رنگ ها را از انگلیسی به فارسی ترجمه کند

فارسی	انگلیسی
زرد	yellow
آبی	blue
قهوه ای	brown
سفید	white
صورتی	pink
نارنجی	orange
سبز	green

این برنامه باید با ساختار switch case نوشته شود. در صورتی که حرف اول رنگ، حرف بزرگ باشد برنامه نیز باید به درستی کار کند