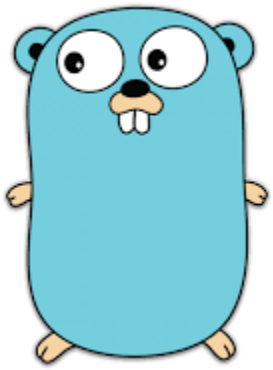


Go

به زبان ساده

تقدیم به

همه جویندگان علم

راه‌های ارتباط با نویسنده

وب سایت: www.w3-farsi.com

لینک تلگرام: https://telegram.me/ebrahimi_younes

ID تلگرام: @ebrahimi_younes

پست الکترونیکی: younes.ebrahimi.1391@gmail.com

GO چیست

زبان برنامه نویسی Go که به گولنگ یا Golang معروف می‌باشد، یک زبان برنامه نویسی است که در سال ۲۰۰۹ توسط Ken Thompson و Rob Pike و Robert Griesemer در شرکت گوگل ابداع و به صورت متن باز منتشر شد.

علاوه بر گوگل، شرکت‌های بزرگی مانند YouTube، BBC، SoundCloud و غیره از این زبان برای طراحی سیستم‌های Back-end استفاده می‌کنند.

می‌توان گفت که Go با ظرافت فراوان قدرت و سرعت زبانی مثل C را با سهولت و سادگی زبانی مثل Python ترکیب کرده. به همین دلیل قادر است طیف بسیار وسیعی از برنامه‌ها را پوشش دهد، از برنامه‌های سیستمی گرفته تا برنامه‌های ساده چند خطی.

Go زبانی است از خانواده C و به همین دلیل برنامه نویسانی که با C++، C، Java، C#، PHP، JavaScript و ... آشنایی دارند، بسیار راحت این زبان را یاد خواهند گرفت.

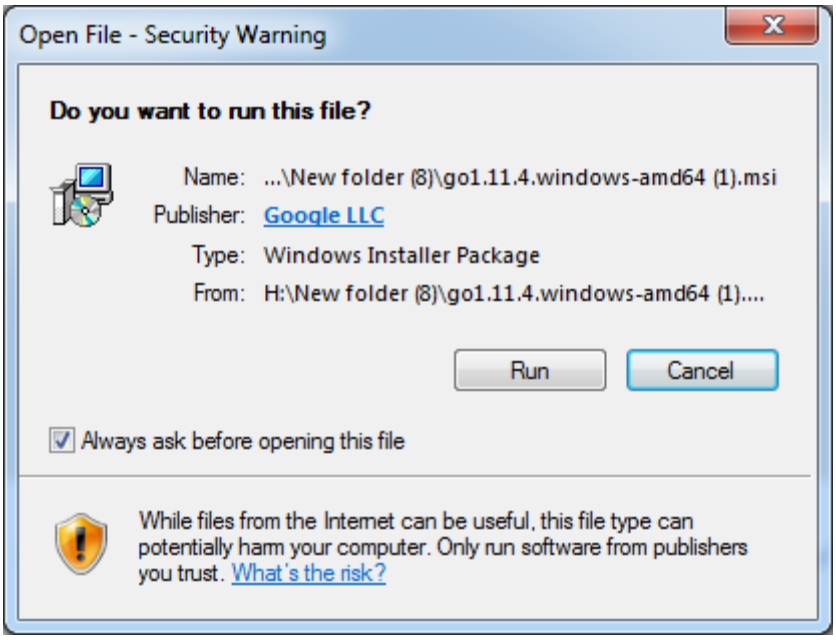
Go یک زبان برنامه‌نویسی همه منظوره با امکانات پیشرفته و دستور زبان شفاف می‌باشد. بخاطر پشتیبانی از گستره بسیاری از پلتفرم‌ها، کتابخانه‌های قدرتمند مستند سازی شده و تمرکز روی اصول مهندسی نرم‌افزار، Go یکی از ایده‌آل ترین زبان‌ها برای یادگیری به عنوان اولین زبان برنامه‌نویسی می‌باشد.

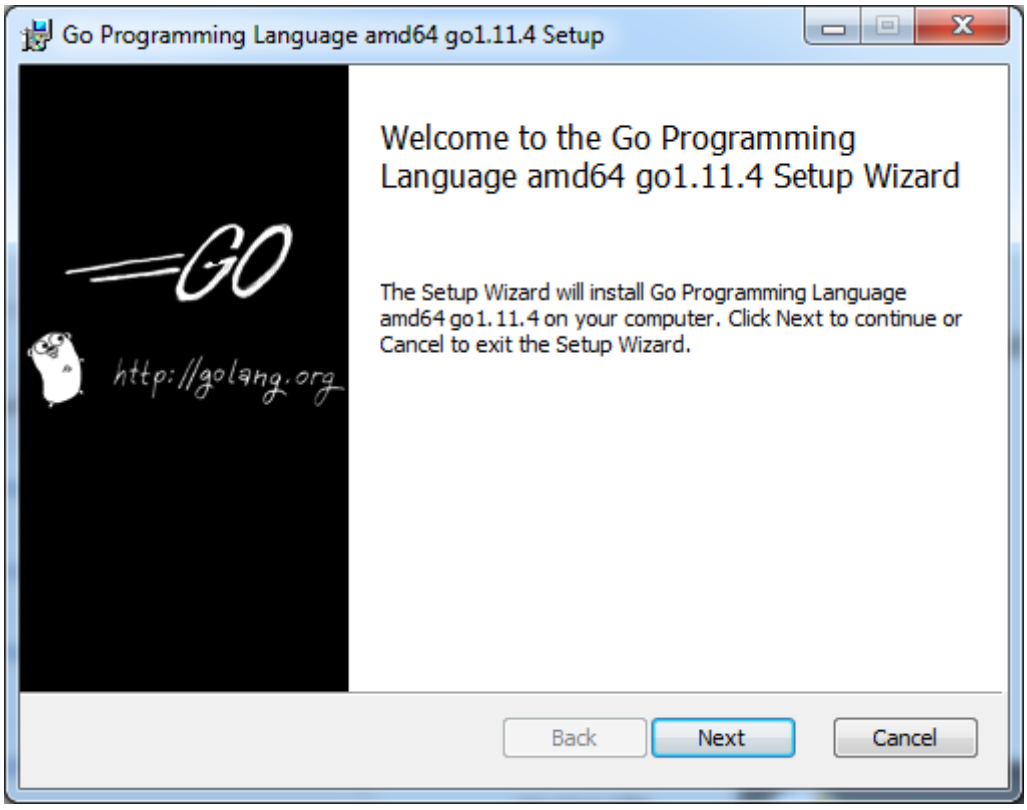
نصب و راه اندازی Go

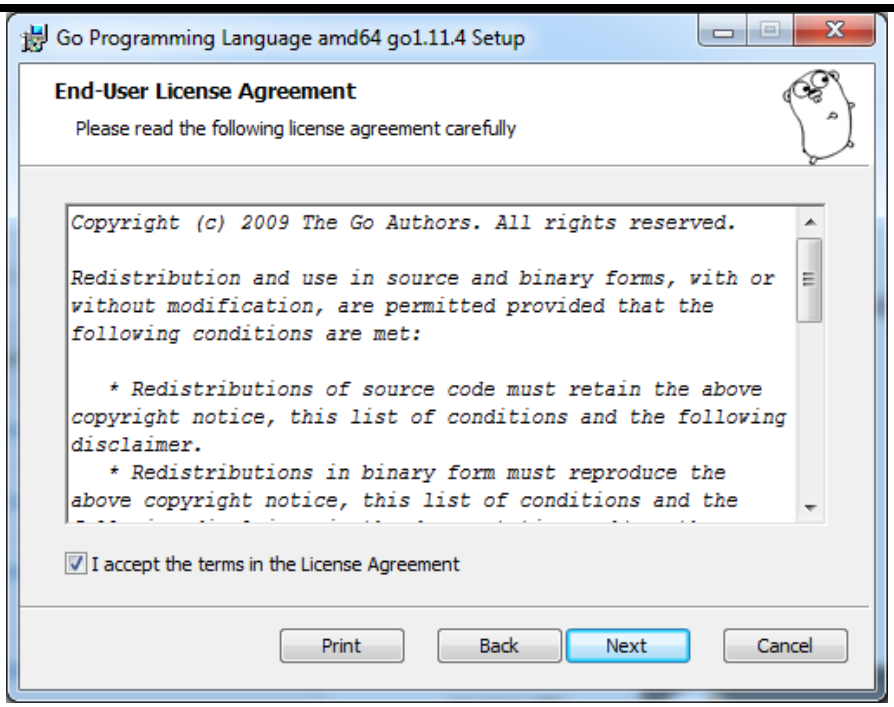
برای کدنویسی به زبان Go و همچنین اجرای کدهای این زبان به دو ابزار احتیاج دارید. یکی از آنها، یک ویرایشگر متن ساده مانند NotePad خود ویندوز و دیگری کامپایلر زبان Go می‌باشد. این کامپایلر را می‌توانید از لینک زیر دانلود کنید:

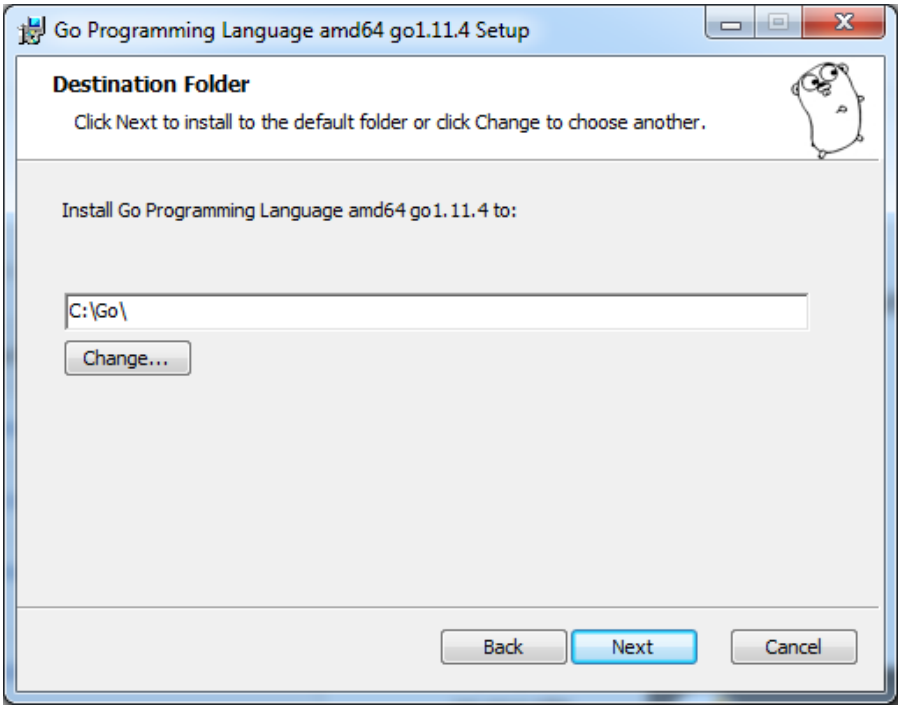
```
http://dl.w3-farsi.com/Software/GO/go1.11.4.windows-amd64.msi
```

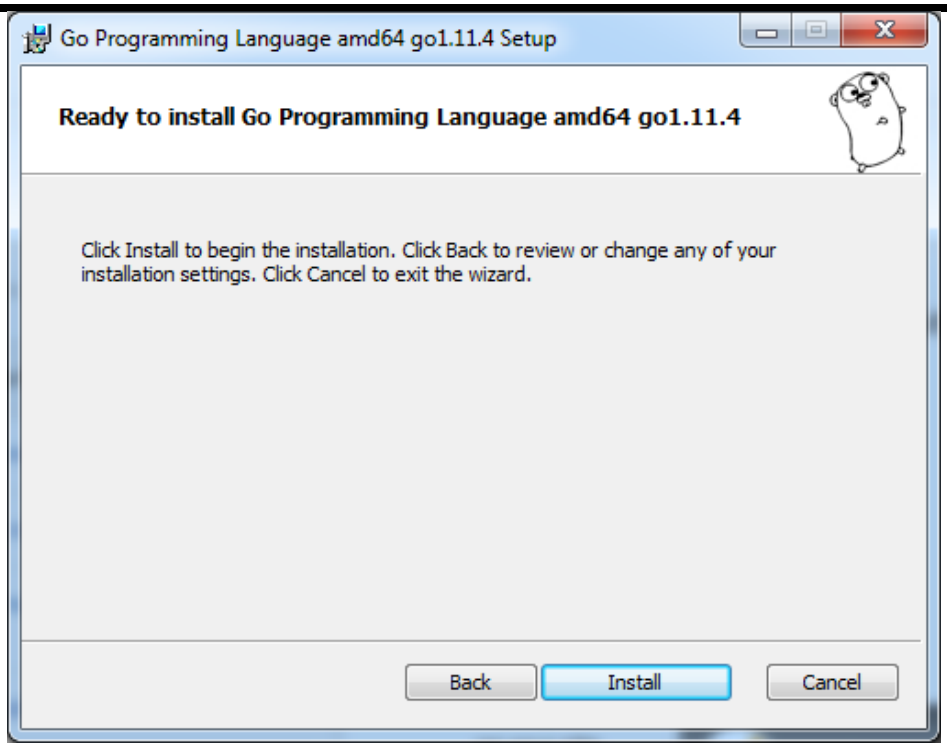
بعد از دانلود فایل بالا، با دوبار کلیک بر روی آن و زدن چند دکمه Next یا Yes، که مراحل آن در زیر نمایش داده شده است، کامپایلر Go نصب می‌شود:

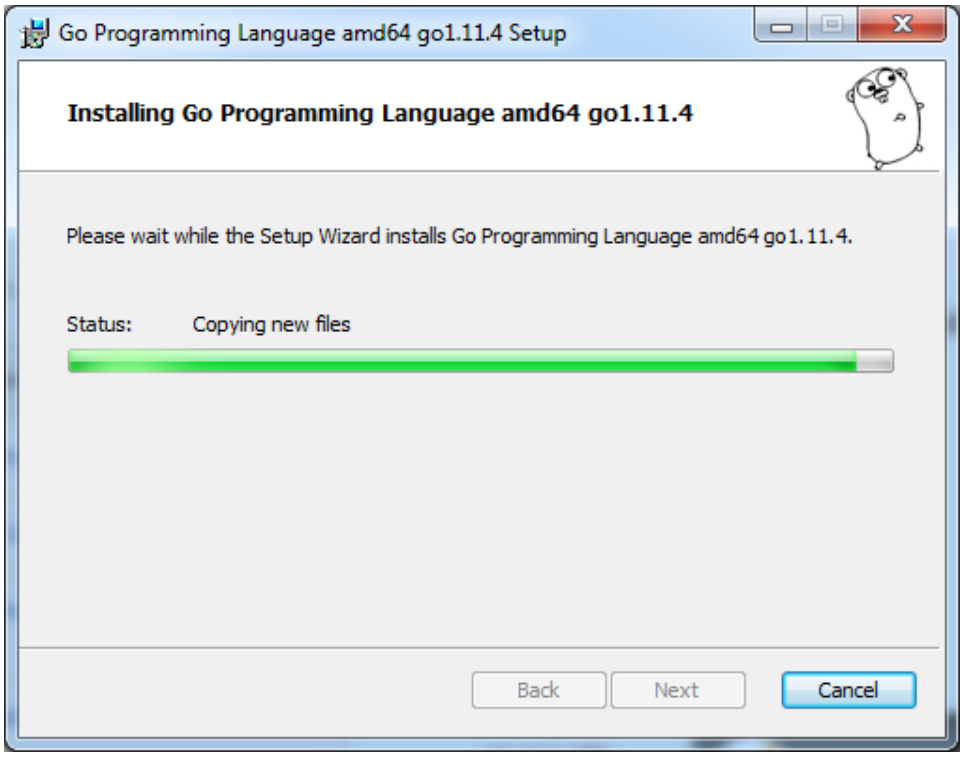

Open File - Security Warning
Do you want to run this file?
Name: ...\New folder (8)\go1.11.4.windows-amd64 (1).msi
Publisher: Google LLC
Type: Windows Installer Package
From: H:\New folder (8)\go1.11.4.windows-amd64 (1)....
Run
Cancel
Always ask before opening this file
While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

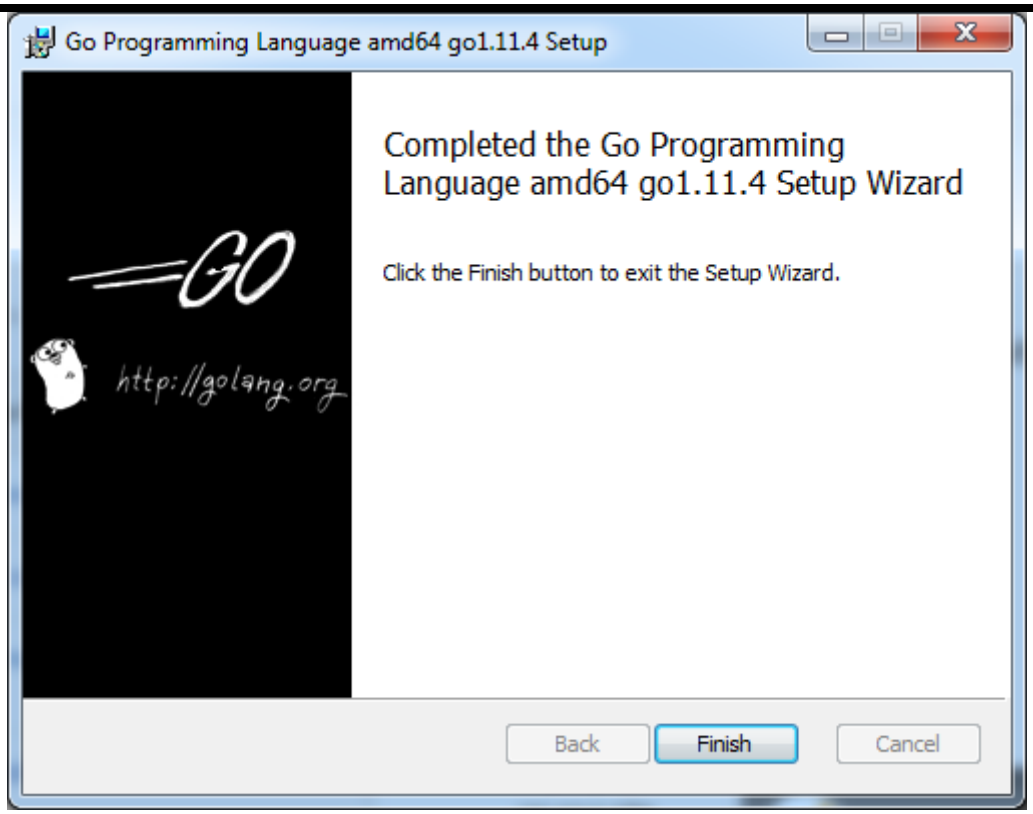

Go Programming Language amd64 go1.11.4 Setup
Welcome to the Go Programming Language amd64 go1.11.4 Setup Wizard
The Setup Wizard will install Go Programming Language amd64 go1.11.4 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.
Back
Next
Cancel


Go Programming Language amd64 go1.11.4 Setup
End-User License Agreement
Please read the following license agreement carefully
Copyright (c) 2009 The Go Authors. All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the
I accept the terms in the License Agreement
Print
Back
Next
Cancel


Go Programming Language amd64 go1.11.4 Setup
Destination Folder
Click Next to install to the default folder or click Change to choose another.
Install Go Programming Language amd64 go1.11.4 to:
C:\Go\
Change...
Back
Next
Cancel


Go Programming Language amd64 go1.11.4 Setup
Ready to install Go Programming Language amd64 go1.11.4
Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.
Back
Install
Cancel


Go Programming Language amd64 go1.11.4 Setup
Installing Go Programming Language amd64 go1.11.4
Please wait while the Setup Wizard installs Go Programming Language amd64 go1.11.4.
Status: Copying new files
Back
Next
Cancel



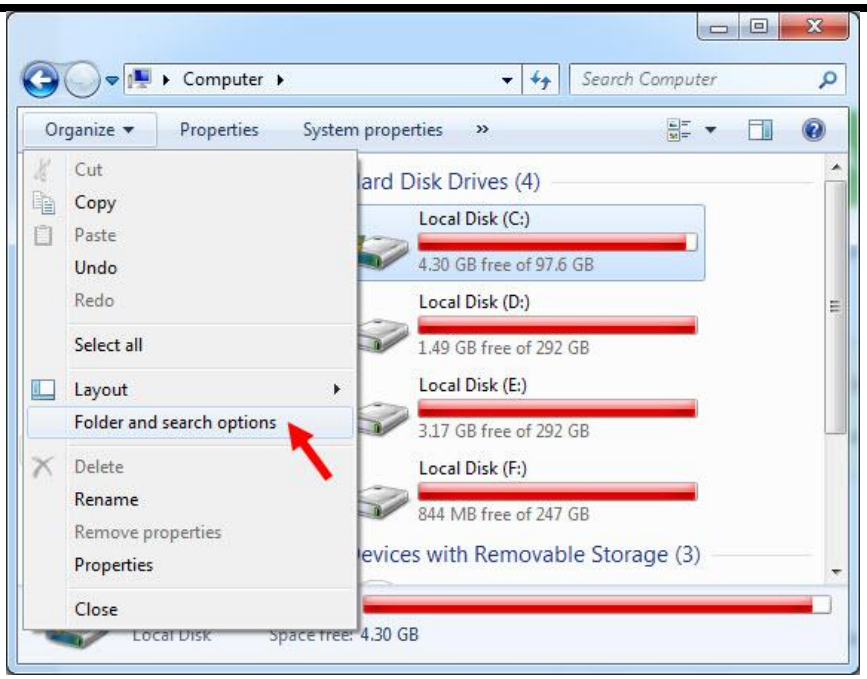
بعد از طی مراحل بالا، شما به راحتی می‌توانید، کدنویسی خود را شروع کنید. در درس بعد شما را با نحوه اجرای کدهای زبان Go آشنا می‌کنم.

ساخت یک برنامه ساده در Go

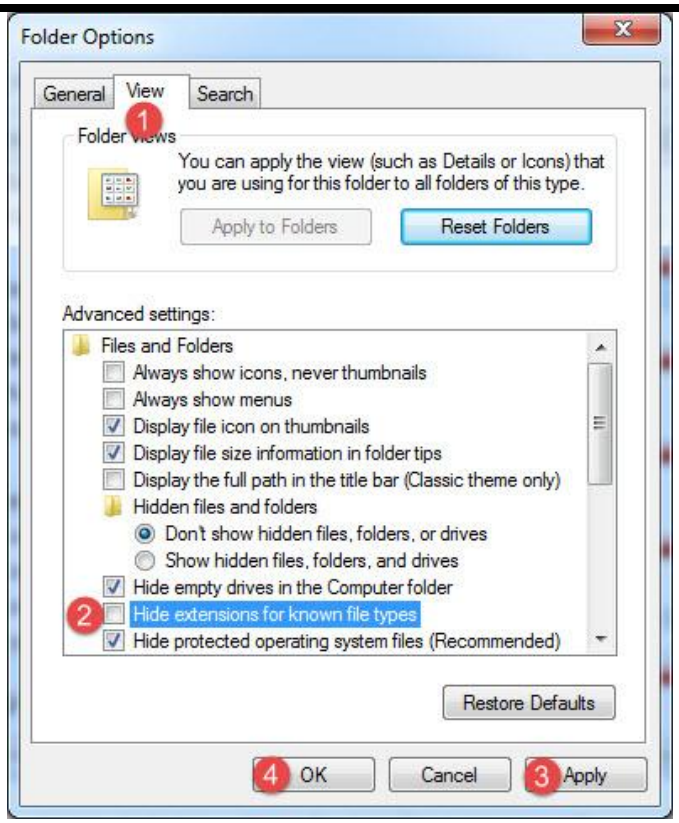
اجازه بدهید یک برنامه بسیار ساده به زبان Go بنویسیم. این برنامه یک پیغام را نمایش می‌دهد. در این درس می‌خواهم ساختار و دستور زبان یک برنامه ساده GO را توضیح دهم. قبل از ایجاد برنامه به این نکته خیلی مهم توجه کنید:

> در نوشتن این برنامه و برنامه‌های آتی، به حروف بزرگ و کوچک، توجه کنید. چون Go به بزرگ و کوچک بودن حروف حساس است.

یکی از تنظیماتی که قبل از شروع این درس توصیه می‌کنیم که اعمال کنید این است که پسوند فایل‌ها داخل ویندوز را قابل مشاهده کنید. برای این کار به `My Computer` رفته و به صورت زیر از منوی `Organize` گزینه `Folder and search options` را بزنید:

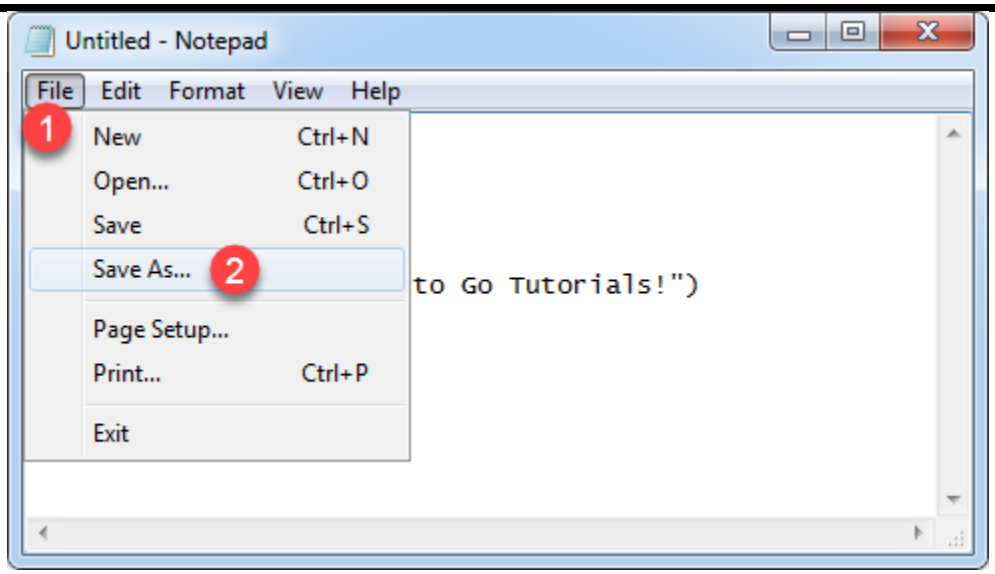


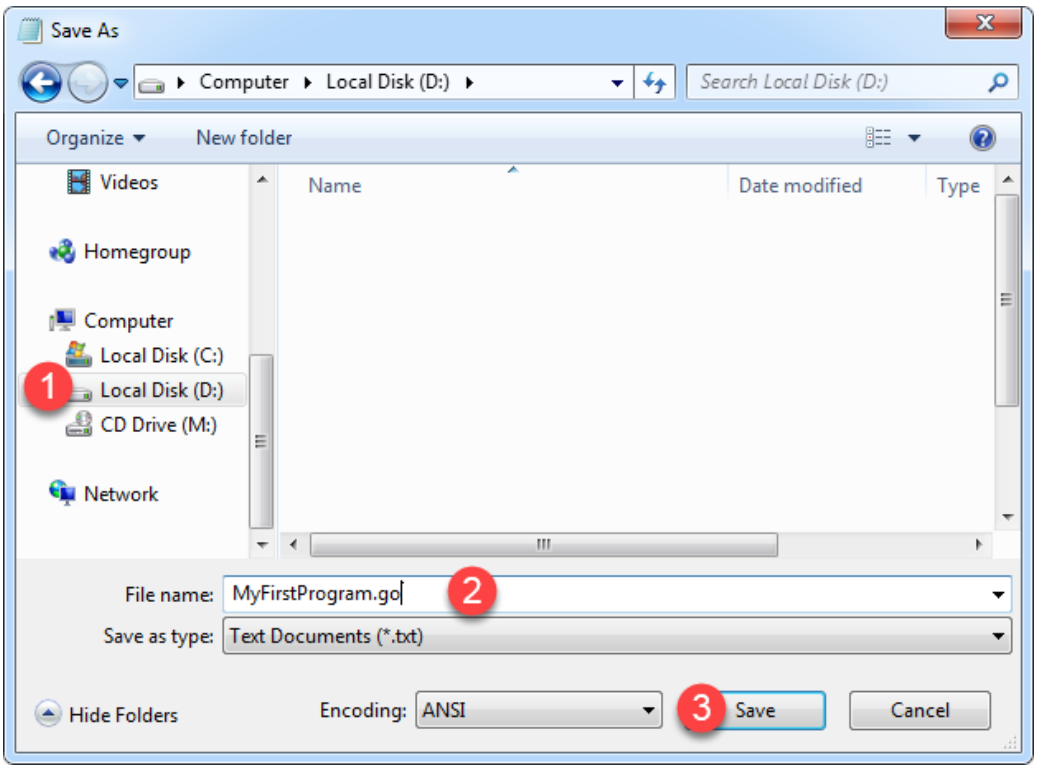
از پنجره باز شده به صورت زیر به سربرگ View رفته و تیک کنار گزینه Hide Extension for khnown file types را بردارید:

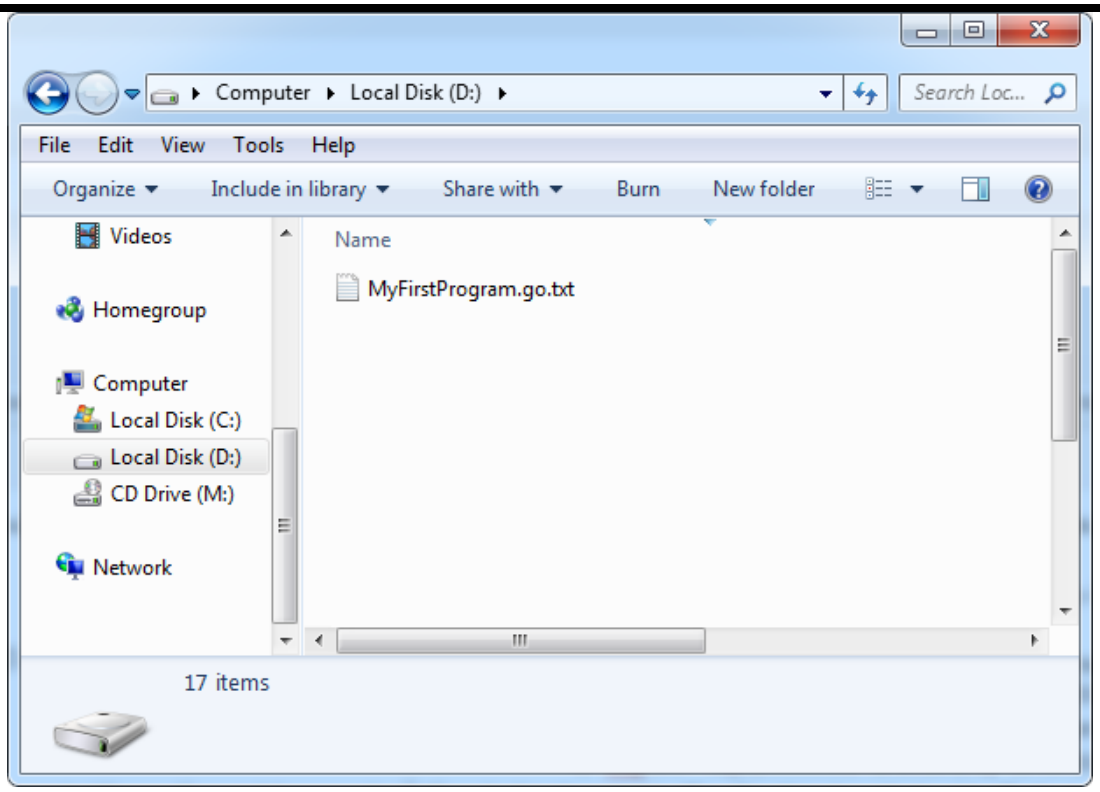


در ادامه شما را نحوه ایجاد اولین برنامه در Go را توضیح می‌دهیم. همانطور که گفته شد، شما برای کامپایل و اجرای برنامه‌های Go به کامپایلر این زبان نیاز دارید، که آن را در درس قبل نصب کردیم و الان فرض می‌کنیم که شما هیچ IDE یا محیط کدنویسی در اختیار ندارید و می‌خواهید یک برنامه Go بنویسید. در این برنامه می‌خواهیم پیغام Welcome to Go Tutorials چاپ شود. ابتدا یک ویرایشگر متن مانند Notepad را باز کرده و کدهای زیر را در داخل آن نوشته (حروف بزرگ و کوچک را رعایت کنید) و با پسوند go ذخیره کنید :

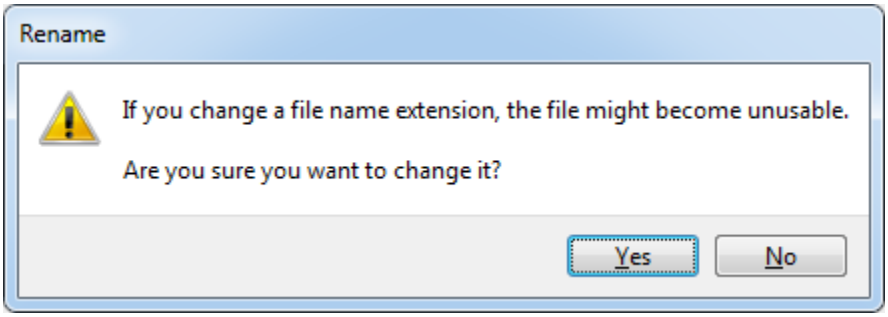
```go
package main

import "fmt"

func main(){
   fmt.Println("Welcome to Go Tutorials!")
}
```


Untitled - Notepad
File Edit Format View Help
1
New Ctrl+N
Open... Ctrl+O
Save Ctrl+S
Save As... 2
Page Setup...
Print... Ctrl+P
Exit
to Go Tutorials!")

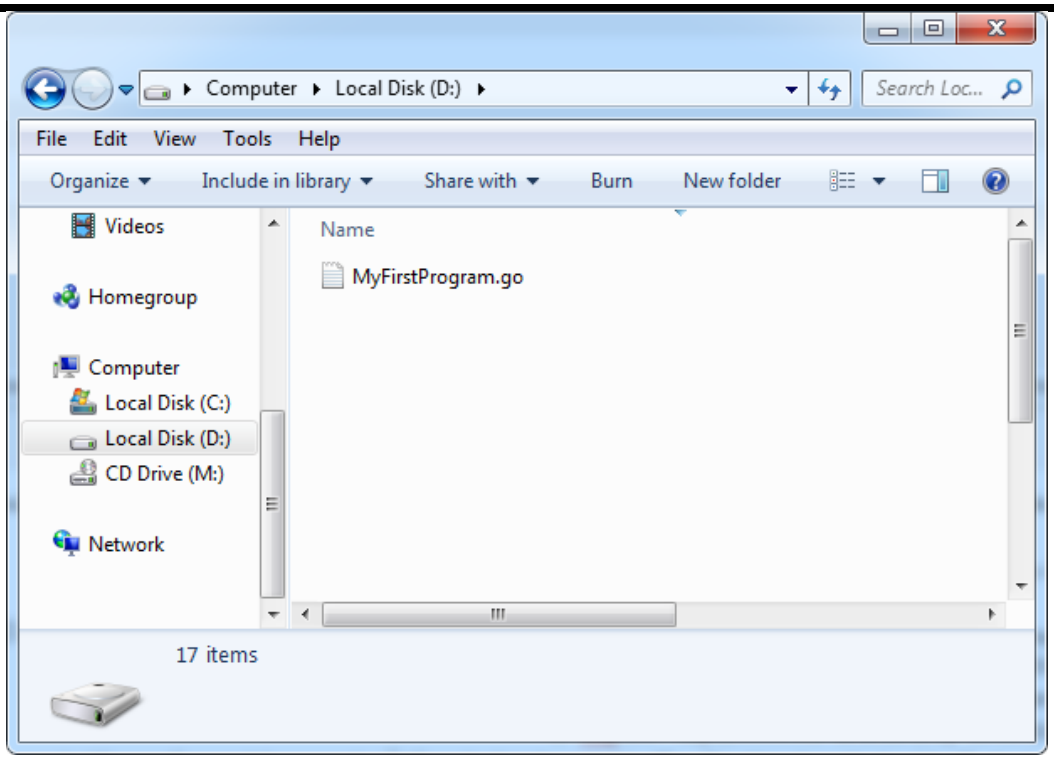

Save As
Computer ▸ Local Disk (D:) ▸
Search Local Disk (D:)
Organize
New folder
Videos
Homegroup
Computer
Local Disk (C:)
1
Local Disk (D:)
CD Drive (M:)
Network
Name
Date modified
Type
File name: MyFirstProgram.go 2
Save as type: Text Documents (*.txt)
Hide Folders
Encoding: ANSI
3 Save
Cancel



همانطور که مشاهده می‌کنید، بعد از ذخیره، فایل با پسوند `MyFirstProgram.go.txt` ذخیره می‌شود که شما باید پسوند `.txt` آن را حذف کنید. هنگام پاک کردن پسوند، پیغامی به صورت زیر ظاهر می‌شود که شما باید بر روی گزینه `Yes` کلیک کنید:



تا شکل نهایی فایل به صورت زیر در آید:



حال نوبت به اجرای برنامه می‌رسد. فایل ما در درایو D قرار دارد. ابتدا cmd را باز کرده و کد زیر را در داخل آن نوشته و دکمه Enter را می‌زنید :

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\siavash>d:

D:\>go run MyFirstProgram.go
Welcome to Go Tutorials!

D:\>
```

همانطور که در کد بالا مشاهده می‌کنید، برای اجرای کدهای Go ابتدا جمله go run و سپس نام پروژه به همراه پسوند آن را می‌نویسیم (مثل MyFirstProgram.go)

ساختار یک برنامه در Go

مثال بالا ساده‌ترین برنامه‌ای است که شما می‌توانید در Go بنویسید. هدف در مثال بالا نمایش یک پیغام در صفحه نمایش است. هر زبان برنامه نویسی دارای قواعدی برای کدنویسی است. اجازه بدهید هر خط کد را در مثال بالا توضیح بدهیم. در خط اول package

تعریف شده است که شامل کدهای نوشته شده توسط شما است و از تداخل نامها جلوگیری می‌کند. درباره package در درسهای آینده توضیح خواهیم داد. فقط این نکته را در همین ابتدا ذکر کنیم که وجود این خط برای اجرای کدها الزامی است.

Go دارای package هایی است که به صورت توکار و هنگام نصب کامپایلر Go نصب می‌شوند و هر کدام برای مقاصد خاصی مورد استفاده قرار می‌گیرند. یکی از این package ها، fmt می‌باشد. برای استفاده از این package ها از کلمه کلیدی import استفاده می‌کنیم (خط ۳). خط ۵ متد ()main یا متد اصلی نامیده می‌شود. هر متد شامل یک سری کد است که وقتی اجرا می‌شوند که متد را صدا بزنیم. درباره متد و نحوه صدا زدن آن در فصول بعدی توضیح خواهیم داد. متد ()main نقطه آغاز اجرای برنامه است. این بدان معناست که ابتدا تمام کدهای داخل متد ()main و سپس بقیه کدها اجرا می‌شود. درباره متد ()main در فصول بعدی توضیح خواهیم داد. متد ()main و سایر متدها دارای آکولاد و کدهایی در داخل آن‌ها می‌باشند و وقتی کدها اجرا می‌شوند که متدها را صدا بزنیم. مثالی از یک خط کد در Go به صورت زیر است :

```
fmt.Println("Welcome to Go Tutorials!")
```

در خط ۵ آکولاد ({) نوشته شده است. آکولاد برای تعریف یک بلوک کد به کار می‌رود. Go یک زبان ساخت یافته است که شامل کدهای زیاد و ساختارهای فراوانی می‌باشد. هر آکولاد باز ({) در Go باید دارای یک آکولاد بسته (}) نیز باشد. همه کدهای نوشته شده از خط ۵ تا خط ۷ یک بلوک کد است. این خط کد پیغام Welcome to Go Tutorials! را در صفحه نمایش نشان می‌دهد. از متد ()Println که در داخل package، fmt قرار دارد، برای چاپ یک رشته استفاده می‌شود. یک رشته گروهی از کاراکترها است که به وسیله دابل کوتیشن (") محصور شده است. مانند "Welcome to Visual Go Tutorials!":

یک کاراکتر می‌تواند یک حرف، عدد، علامت یا .... باشد. در کل مثال بالا نحوه استفاده از متد ()Println نشان داده شده است. این متد یک متد از پکیج fmt بوده و از آن برای چاپ مقادیر استفاده می‌شود. Go فضای خالی و خطوط جدید را نادیده می‌گیرد. همیشه به یاد داشته باشید که Go به بزرگی و کوچکی حروف حساس است. یعنی به طور مثال MAN و man در Go با هم فرق دارند. رشته‌ها و توضیحات از این قاعده مستثنی هستند که در درسهای آینده توضیح خواهیم داد. مثلاً کدهای زیر با خطا مواجه می‌شوند و اجرا نمی‌شوند :

```
fmt.println("Welcome to Go Tutorials!")
fmt.PRIintln("Welcome to Go Tutorials!")
FMT.Println("Welcome to Go Tutorials!")
```

تغییر در بزرگی و کوچکی حروف از اجرای کدها جلوگیری می‌کند. اما کد زیر کاملاً بدون خطا است :

```
fmt.Println("Welcome to Go Tutorials!")
```

همیشه کدهای خود را در داخل آکولاد بنویسید.

```
{
   statement1
}
```

این کار باعث می‌شود که کدنویسی شما بهتر به چشم بیاید و تشخیص خطاها راحت‌تر باشد. در زبان Go نیازی به سمیکالن (;) ندارید. همین که بعد از هر خط یک بار Enter بزنید به منزله این است که آن خط به پایان رسیده است. مثلاً دو خط زیر دو دستور جدا هستند:

```
fmt.Println("Welcome to Go Tutorials!")
fmt.Println("Welcome to Go Tutorials!")
```

توضیحات

وقتی که کدی تایپ می کنید شاید بخواهید که متنی جهت یادآوری وظیفه آن کد به آن اضافه کنید. در Go (و بیشتر زبانهای برنامه نویسی) می توان این کار را با استفاده از توضیحات انجام داد. توضیحات متونی هستند که توسط کامپایلر نادیده گرفته می شوند و به عنوان بخشی از کد محسوب نمی شوند.

هدف اصلی از ایجاد توضیحات، بالا بردن خوانایی و تشخیص نقش کدهای نوشته شده توسط شما، برای دیگران است. فرض کنید که می خواهید در مورد یک کد خاص، توضیح بدهید، می توانید توضیحات را در بالای کد یا کنار آن بنویسید. از توضیحات برای مستند سازی برنامه هم استفاده می شود. در برنامه زیر نقش توضیحات نشان داده شده است :

```
1  package main
2
3  import "fmt"
4
5  func main(){
6     //This line will print the message hello world
7     fmt.Println("Hello World!")
8  }
```

در کد بالا، خط ۶ کد بالا یک توضیح درباره خط ۷ است که به کاربر اعلام می کند که وظیفه خط ۷ چیست ؟ با اجرای کد بالا فقط جمله Hello World چاپ شده و خط اول در خروجی نمایش داده نمی شود چون کامپایلر توضیحات را نادیده می گیرد. همانطور که مشاهده می کنید برای درج توضیحات در Go از علامت // استفاده می شود. اگر توضیح درباره یک کد به بیش از یک خط نیاز باشد از توضیحات

چند خطی استفاده می‌شود. توضیحات چند خطی با */ شروع و با /* پایان می‌یابند. هر نوشته‌ای که بین این دو علامت قرار بگیرد جز توضیحات محسوب می‌شود.

```
package main

import "fmt"

func main(){
   /*This line will print
   the message hello world*/
   fmt.Println("Hello World!")
}
```

کاراکترهای کنترلی

کاراکترهای کنترلی کاراکترهای ترکیبی هستند که با یک بک اسلش (\) شروع می‌شوند و به دنبال آنها یک حرف یا عدد می‌آید و یک رشته را با فرمت خاص نمایش می‌دهند. برای مثال برای ایجاد یک خط جدید و قرار دادن رشته در آن می‌توان از کاراکتر کنترلی \n استفاده کرد :

```
fmt.Println("Hello\nWorld!")
```

```
Hello
World
```

مشاهده کردید که کامپایلر بعد از مواجهه با کاراکتر کنترلی \n نشانگر ماوس را به خط بعد برده و بقیه رشته را در خط بعد نمایش می‌دهد. متد Println() هم مانند کاراکتر کنترلی \n یک خط جدید ایجاد می‌کند، البته بدین صورت که در انتهای رشته یک کاراکتر کنترلی \n اضافه می‌کند :

```
fmt.Println("Hello World!")
```

کد بالا و کد زیر هیچ فرقی با هم ندارند :

```
fmt.Print("Hello World!\n")
```

متد Print() کارکردی شبیه به Println() دارد با این تفاوت که نشانگر ماوس را در همان خط نگه می‌دارد و خط جدید ایجاد نمی‌کند.

جدول زیر لیست کاراکترهای کنترلی و کارکرد آنها را نشان می‌دهد :

عملکرد	کاراکتر کنترلی	عملکرد	کاراکتر کنترلی
Form Feed	\f	چاپ کوتیشن	\’
خط جدید	\n	چاپ دابل کوتیشن	\”
سر سطر رفتن	\r	چاپ بک اسلش	\\
حرکت به صورت افقی	\t	چاپ فضای خالی	\0
حرکت به صورت عمودی	\v	صدای بیپ	\a
چاپ کاراکتر یونیکد	\u	حرکت به عقب	\b

ما برای استفاده از کاراکترهای کنترلی از بک اسلش (\) استفاده می‌کنیم. از آنجاییکه \ معنای خاصی به رشته‌ها می‌دهد برای چاپ بک اسلش (\) باید از (\\) استفاده کنیم :

```
fmt.Println("We can print a \\ by using the \\\\ escape sequence.")
```

```
We can print a \ by using the \\ escape sequence.
```

یکی از موارد استفاده از \\، نشان دادن مسیر یک فایل در ویندوز است :

```
fmt.Println("C:\\Program Files\\Some Directory\\SomeFile.txt")
```

```
C:\Program Files\Some Directory\SomeFile.txt
```

از آنجاییکه از دابل کوتیشن (“) برای نشان دادن رشته‌ها استفاده می‌کنیم برای چاپ آن از \” استفاده می‌کنیم :

```
fmt.Println("I said, \"Motivate yourself!\".")
```

```
I said, "Motivate yourself!".
```

همچنین برای چاپ کوتیشن (‘) از \’ استفاده می‌کنیم :

```
fmt.Println("The programmer\'s heaven.")
```

```
The programmer's heaven.
```

برای ایجاد فاصله بین حروف یا کلمات از \t استفاده می‌شود :

```
fmt.Println("Left\tRight")
```

```
Left    Right
```

هر تعداد کاراکتر که بعد از کاراکتر کنترلی \r۳ بیایند به اول سطر منتقل و جایگزین کاراکترهای موجود می‌شوند :

```
fmt.Println("Mitten\rK")
```

```
K
```

مثلاً در مثال بالا کاراکتر K بعد از کاراکتر کنترلی \r۳ آمده است. کاراکتر کنترلی حرف K را به ابتدای سطر برده و جایگزین حرف M می‌کند.

برای چاپ کاراکترهای یونیکد می‌توان از \u استفاده کرد. برای استفاده از \u، مقدار در مبنای ۱۶ کاراکتر را درست بعد از علامت \u قرار می‌دهیم. برای مثال اگر بخواهیم علامت کپی رایت (©) را چاپ کنیم، باید بعد از علامت \u مقدار A900 را قرار دهیم مانند :

```
fmt.Println("\u00A9")
```

```
©
```

برای مشاهده لیست مقادیر مبنای ۱۶ برای کاراکترهای یونیکد به لینک زیر مراجعه نمایید :

http://www.ascii.cl/htmlcodes.htm

اگر کامپایلر به یک کاراکتر کنترلی غیر مجاز برخورد کند، برنامه پیغام خطا می‌دهد. بیشترین خطا زمانی اتفاق می افتد که برنامه نویس برای چاپ اسلش (\) از \\ استفاده می‌کند. برای دریافت اطلاعات بیشتر در مورد کاراکترهای کنترلی به لینک زیر مراجعه کنید :

https://msdn.microsoft.com/en-us/library/h21280bw.aspx

متغیر

متغیر مکانی از حافظه است که شما می‌توانید مقادیری را در آن ذخیره کنید. می‌توان آن را به عنوان یک ظرف تصور کرد که داده‌های خود را در آن قرار داده‌اید. محتویات این ظرف می‌تواند پاک شود یا تغییر کند. هر متغیر دارای یک نام نیز هست. که از طریق آن می‌توان متغیر را از دیگر متغیرها تشخیص داد و به مقدار آن دسترسی پیدا کرد. همچنین دارای یک مقدار می‌باشد که می‌تواند توسط کاربر انتخاب شده باشد یا نتیجه یک محاسبه باشد. مقدار متغیر می‌تواند تهی نیز باشد. متغیر دارای نوع نیز هست بدین معنی که نوع آن با نوع داده‌ای که در آن ذخیره می‌شود یکی است.

متغیر دارای عمر نیز هست که از روی آن می‌توان تشخیص داد که متغیر باید چقدر در طول برنامه مورد استفاده قرار گیرد. و در نهایت متغیر دارای محدوده استفاده نیز هست که به شما می‌گوید که متغیر در چه جای برنامه برای شما قابل دسترسی است. ما از متغیرها به عنوان یک انبار موقتی برای ذخیره داده استفاده می‌کنیم. هنگامی که یک برنامه ایجاد می‌کنیم احتیاج به یک مکان برای ذخیره داده، مقادیر یا داده‌هایی که توسط کاربر وارد می‌شوند، داریم. این مکان، همان متغیر است.

برای این از کلمه متغیر استفاده می‌شود چون ما می‌توانیم بسته به نوع شرایط هر جا که لازم باشد، مقدار آن را تغییر دهیم. متغیرها موقتی هستند و فقط موقعی مورد استفاده قرار می‌گیرند که برنامه در حال اجراست و وقتی شما برنامه را می‌بندید محتویات متغیرها نیز پاک می‌شود. قبلاً ذکر شد که به وسیله نام متغیر می‌توان به آن دسترسی پیدا کرد. برای نامگذاری متغیرها باید قوانین زیر را رعایت کرد :

- **نام متغیر باید با یکی از حروف الفبا** `(a-z or A-Z)` **یا علامت _ شروع شود.**
- **نمی‌تواند شامل کاراکترهای غیرمجاز مانند** `., $, ^, ?, #` **باشد.**
- **نمی‌توان از کلمات رزرو شده در** `Go` **برای نام متغیر استفاده کرد.**
- **نام متغیر نباید دارای فضای خالی** `(spaces)` **باشد.**
- **اسامی متغیرها نسبت به بزرگی و کوچکی حروف حساس هستند. در** `Go` **دو حرف مانند** `a` **و** `A` **دو کاراکتر مختلف به حساب می‌آیند.**

دو متغیر با نامهای `myNumber` و `MyNumber` دو متغیر مختلف محسوب می‌شوند چون یکی از آنها با حرف کوچک `m` و دیگری با حرف بزرگ `M` شروع می‌شود. متغیر دارای نوع هست که نوع داده‌ای را که در خود ذخیره می‌کند را نشان می‌دهد. در درس بعد در مورد انواع داده ها در Go توضیح می دهیم. لیست کلمات کلیدی Go، که نباید از آنها در نامگذاری متغیرها استفاده کرد در زیر آمده است:

```
break        default      func         interface    select
case         defer        go           map          struct
chan         else         goto         package      switch
const        fallthrough  if           range        type
continue     for          import       return       var
```

انواع داده

متغیرها در Go می‌توانند انواع مختلف داده را در خود ذخیره کنند. این داده‌ها دارای مجموعه مشخصی از مقادیر شامل اعداد، کاراکترها و یا مقادیر بولی هستند. در جدول زیر انواع داده و محدوده آنها آمده است :

نوع	توضیح
Boolean (منطقی)	این نوع از داده شامل دو مقدار منطقی true و false می‌باشد.
Numeric (عددی)	این نوع، شامل تمامی اعداد صحیح و اعشار می‌باشد.
String (رشته‌ای)	این نوع شامل مجموعه‌ای از کارکترها می‌باشد.

نوع عددی در جدول بالا، همانطور که گفته شد شامل همه اعداد می‌شود. اعداد صحیحی که در این گروه هستند، در جدول زیر آمده‌اند:

نوع	دامنه
uint8	اعداد صحیح بین ۰ تا ۲۵۵
uint16	اعداد صحیح بین ۰ تا ۶۵۵۳۵
uint32	اعداد صحیح بین ۰ تا ۴۲۹۴۹۶۷۲۹۵
uint64	اعداد صحیح بین ۰ تا ۱۸۴۴۶۷۴۴۰۷۳۷۰۹۵۵۱۶۱۵
int8	اعداد صحیح بین ۱۲۸- تا ۱۲۷
int16	اعداد صحیح بین ۳۲۷۶۸- تا ۳۲۷۶۷
int32	اعداد صحیح بین ۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷
int64	اعداد صحیح بین ۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۵۸۰۸- تا ۹۲۲۳۳۷

استفاده از متغیرها

در مثال زیر نحوه تعریف و مقدار دهی متغیرها نمایش داده شده است :

```go
package main

import "fmt"

func main() {

    //Declare variables
    var num1    int
    var num2    int
    var num3    float32
    var num4    float32
    var boolVal bool
    var myChar  string

    //Assign values to variables
    num1    = 1
    num2    = 2
    num3    = 3.54
    num4    = 4.12
    boolVal = true
    myChar  = "R"

    //Show the values of the variables
    fmt.Printf("num1    = %d   \n", num1   )
    fmt.Printf("num3    = %.2f \n", num3   )
    fmt.Printf("num4    = %.2f \n", num4   )
    fmt.Printf("boolVal = %t   \n", boolVal)
    fmt.Printf("num2    = %d   \n", num2   )
    fmt.Printf("myChar  = %s   \n", myChar )
}
```

```
num1    =  1
num3    =  3.54
num4    =  4.12
boolVal =  true
num2    =  2
myChar  =  "R"
```

تعریف متغیر

برای تعریف متغیر از دو کلمه var به صورت زیر استفاده می‌شود:

```go
var identifier type
```

var یک کلمه کلیدی، identifier نام و type نوع متغیر است. در این روش ما صراحتاً نوع متغیر را ذکر می‌کنیم. دو روش دیگر هم برای تعریف متغیر وجود دارد که در زیر به آنها اشاره شده است :

```
var identifier = value

//or

identifier := value
```

در هر دو روش بالا، باید فوراً بعد از علامت مساوی مقدار متغیر را هم بنویسیم، تا کامپایلر به صورت خودکار و با توجه به مقدار نوع متغیر را تشخیص دهد ولی در روش اول همانطور که در کد ابتدای درس می‌بینیم، ما اول متغیرها را در خطوط ۱۳-۸ تعریف و سپس در خطوط ۲۱-۱۶ مقداردهی کرده‌ایم. پس خطوط ۲۱-۸ کد بالا را می‌توان به صورت زیر هم خلاصه کرد:

```
var num1    =  1
var num2    =  2
var num3    =  3.54
var num4    =  4.12
var boolVal =  true
var myChar  =  "R"
```

یا

```
num1    :=  1
num2    :=  2
num3    :=  3.54
num4    :=  4.12
boolVal :=  true
myChar  :=  "R"
```

تعریف متغیر با مقدار دهی متغیرها متفاوت است. تعریف متغیر یعنی انتخاب نوع و نام برای متغیر (خطوط ۱۳-۸) ولی مقدار دهی یعنی اختصاص یک مقدار به متغیر (خطوط ۲۱-۱۶). یک رو دیگر برای تعریف چند متغیر به صورت زیر است:

```
var (
    num1    =  1
    num2    =  2
    num3    =  3.54
    num4    =  4.12
    boolVal =  true
    myChar  =  "R"
)
```

نامگذاری متغیرها

- **نام متغیر باید با یک حرف یا زیرخط و به دنبال آن حرف یا عدد شروع شود.**

- **نمی‌توان از کاراکترهای خاص مانند #، %، & یا عدد برای شروع نام متغیر استفاده کرد مانند :** `2numbers`
- **نام متغیر نباید دارای فاصله باشد. برای نام‌های چند حرفی می‌توان به جای فاصله از علامت زیرخط یا _ استفاده کرد.**

نامهای مجاز :

```
num1  myNumber  studentCount  total       first_name    _minimum
num2  myChar    average      amountDue   last_name     _maximum
name  counter   sum          isLeapYear  color_of_car  _age
```

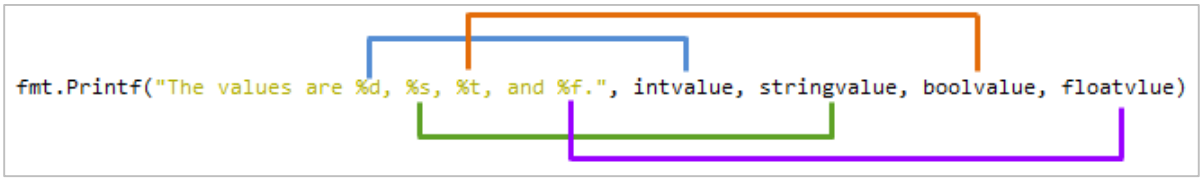
نامهای غیر مجاز :

```
123        #numbers#  #ofstudents  1abc2
123abc     $money     first name   ty.np
my number  this&that  last name    1:00
```

اگر به نامهای مجاز در مثال بالا توجه کنید متوجه قراردادهای به کار رفته در نامگذاری آن‌ها خواهید شد. یکی از روشهای نامگذاری، نامگذاری کوهان شتری است. در این روش که برای متغیرهای دو کلمه‌ای به کار می‌رود، اولین حرف اولین کلمه با حرف کوچک و اولین حرف دومین کلمه با حرف بزرگ نمایش داده می‌شود مانند : `myNumber` توجه کنید که اولین حرف کلمه `Number` با حرف بزرگ شروع شده است. مثال دیگر کلمه `numberOfStudents` است. اگر توجه کنید، بعد از اولین کلمه، حرف اول سایر کلمات با حروف بزرگ نمایش داده شده است.

به متد `Printf()` در خطوط (۲۹-۲۴) توجه کنید. این متد دو آرگومان قبول می‌کند. آرگومان‌ها اطلاعاتی هستند که متد با استفاده از آنها کاری انجام می‌دهد. آرگومان‌ها به وسیله کاما از هم جدا می‌شوند. آرگومان اول یک رشته قالب بندی شده است و آرگومان دوم مقداری است که توسط رشته قالب بندی شده مورد استفاده قرار می‌گیرد. اگر به دقت نگاه کنید، در داخل رشته قالب بندی شده، علامت % به همراه کاراکترهایی آمده است. این ترکیب (کاراکتر و %) با مقدار آرگومان بعد از رشته جایگزین می‌شوند. برای درک بهتر، به مثال زیر که در آن از ۴ جانگهدار استفاده شده است توجه کنید:

```
fmt.Printf("The values are %d, %s, %t, and %f.", intvalue, stringvalue, boolvalue, floatvlue)
```



اما از کاراکترها چه زمانی باید استفاده شود؟ از کاراکتر d به همراه % برای نمایش اعداد صحیح، از s برای رشته، از t برای مقادیر منطقی یعنی true و false و از f برای نمایش اعداد اعشاری استفاده می‌شود. ممکن است که این سؤال برایتان پیش آمده باشد که معنای %.2f چیست؟ بدین معناست که عدد را تا دو رقم بعد از اعشار نمایش بده.

ثابت

ثابت‌ها، انواعی از متغیرها هستند که مقدار آنها در طول برنامه تغییر نمی‌کند. ثابت‌ها حتماً باید مقدار دهی اولیه شوند و اگر مقدار دهی آنها فراموش شود در برنامه خطا به وجود می‌آید. بعد از این که به ثابت‌ها مقدار اولیه اختصاص داده شد هرگز در زمان اجرای برنامه نمی‌توان آن را تغییر داد. برای تعریف ثابت‌ها باید از کلمه کلیدی const استفاده کرد. معمولاً نام ثابت‌ها را طبق قرارداد با حروف بزرگ می‌نویسند تا تشخیص آنها در برنامه راحت باشد. نحوه تعریف ثابت در زیر آمده است :

```
const identifier data_type = initial_value
```

مثال :

```
func main() {
    const NUMBER int = 1

    NUMBER = 10 //ERROR, Cant modify a constant
}
```

در این مثال می‌بینید که مقدار دادن به یک ثابت، که قبلاً مقدار دهی شده برنامه را با خطا مواجه می‌کند. نکته‌ی دیگری که نباید فراموش شود این است که نباید مقدار ثابت را با مقدار دیگر متغیرهای تعریف شده در برنامه برابر قرار داد.مثال :

```
var   someVariable int

const MY_CONST     int = someVariable
```

ممکن است این سؤال برایتان پیش آمده باشد که دلیل استفاده از ثابت‌ها چیست؟ اگر مطمئن هستید که مقادیری در برنامه وجود دارند که هرگز در طول برنامه تغییر نمی‌کنند بهتر است که آنها را به صورت ثابت تعریف کنید. این کار هر چند کوچک کیفیت برنامه شما را بالا می‌برد.

تبدیل انواع داده

توصیه می کنیم قبل از شروع این درس ابتدا دو مطلب زیر را بخوانید :

```
http://www.w3-farsi.com/?p=5698
http://www.w3-farsi.com/?p=5710
```

تبدیل نوع یا Type casting، روشی برای تبدیل نوع یک متغیر به نوع دیگر است. نحوه Type casting در Go به صورت زیر است:

```
type_name(expression)
```

type_name نوع جدید و expression نام متغیری است که قرار است به نوع دیگر تبدیل شود. به کد زیر توجه کنید:

```
1  package main
2
3  import "fmt"
4
5  func main() {
6    var number1 byte = 5
7    var number2 int  = int(number1)
8
9    fmt.Println(number2)
10 }
```

```
5
```

در خط ۷ کد بالا بعد از نوع جدید، نام متغیر موجود در خط ۶ را ذکر کرده ایم. این بدین معناست که متغیر number1 را به نوع int تبدیل کن و در داخل متغیر number2 قرار بده. در مثال نوع داده‌ای byte می‌تواند مقادیر ۰ تا ۲۵۵ را در خود ذخیره کند و نوع داده‌ای int مقادیر ۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷ را شامل می‌شود. پس می‌توانید بدون هیچ مشکلی یک متغیر از نوع byte را به یک نوع int تبدیل کنید. مقدار ۵ متغیر number1 در محدوده مقادیر byte یعنی اعداد بین ۲۵۵-۰ قرار دارد اما متغیری از نوع بایت حافظه کمتری نسبت به متغیری از نوع صحیح اشغال می‌کند. نوع byte شامل ۸ بیت یا ۸ رقم دودویی است در حالی که نوع int شامل ۳۲ بیت یا رقم باینری است. یک عدد باینری عددی متشکل از ۰ و ۱ است. برای مثال عدد ۵ در کامپیوتر به عدد باینری ۱۰۱ ترجمه می‌شود. بنابراین وقتی ما عدد ۵ را در یک متغیر از نوع بایت ذخیره می‌کنیم عددی به صورت زیر نمایش داده می‌شود :

```
00000101
```

و وقتی آن را در یک متغیر از نوع صحیح ذخیره می‌کنیم به صورت زیر نمایش داده می‌شود :

```
00000000000000000000000000000101
```

بنابراین قرار دادن یک مقدار int، در یک متغیر byte، درست مانند این است که ما سعی کنیم که یک توپ فوتبال را در یک سوراخ کوچک گلف جای دهیم. حال به مثال زیر توجه کنید :

```go
package main

import "fmt"

func main() {
  var num1 int   = 300
  var num2 byte  = byte(num1)

  fmt.Println(num2)
}
```

```
44
```

در کد بالا یک num1 یک نوع صحیح با مقدار ۳۰۰ است که آن را در خط ۷ به نوع byte تبدیل کرده و در متغیر num2 قرار داده ایم. پس چرا خروجی ۴۴ است؟ دلیل آن این است که نوع byte اعداد بین ۰-۲۵۵ را در خود ذخیره می کند و عدد ۳۰۰ جز این محدود نیست. اما چرا عدد ۴۴؟ این کار به تعداد بیتها بستگی دارد. یک byte دارای ۸ بیت است درحالی که int دارای ۳۲ بیت است. حال اگر به مقدار باینری ۲ عدد توجه کنید متوجه می‌شوید که چرا خروجی عدد ۴۴ است.

```
300 = 00000000000000000000000100101100
255 =                         11111111
-------------------------------------
 44 =                         00101100
```

خروجی بالا نشان می‌دهد که بیشترین مقدار byte که عدد ۲۵۵ است می‌تواند فقط شامل ۸ بیت باشد (۱۱۱۱۱۱۱۱) بنابراین فقط ۸ بیت اول مقدار int به متغیر byte انتقال می‌یابد که شامل (۰۰۱۰۱۱۰۰) یا عدد ۴۴ در مبنای ۱۰ است.

عبارات و عملگرها

ابتدا با دو کلمه آشنا شوید :

- **عملگر: نمادهایی هستند که اعمال خاص انجام می‌دهند.**
- **عملوند: مقادیری که عملگرها بر روی آنها عملی انجام می‌دهند.**

مثلاً X+Y: یک عبارت است که در آن X و Y عملوند و علامت + عملگر به حساب می‌آیند.

زبانهای برنامه نویسی جدید دارای عملگرهایی هستند که از اجزاء معمول زبان به حساب می‌آیندGO . دارای عملگرهای مختلفی از جمله عملگرهای ریاضی، تخصیصی، مقایسه‌ای، منطقی، بیتی و متفرقه می‌باشد. از عملگرهای ساده ریاضی می‌توان به عملگر جمع و تفریق اشاره کرد. دو نوع عملگر در Go وجود دارد :

- **یگانی** - (Unary) **به یک عملوند نیاز دارد**
- **دودویی** - (Binary) **به دو عملوند نیاز دارد**

انواع مختلف عملگر که در این بخش مورد بحث قرار می‌گیرند عبارتند از :

- **عملگرهای ریاضی**
- **عملگرهای تخصیصی**
- **عملگرهای مقایسه‌ای**
- **عملگرهای منطقی**
- **عملگرهای بیتی**
- **عملگرهای متفرقه**

عملگرهای ریاضی

Go از عملگرهای ریاضی برای انجام محاسبات استفاده می‌کند. جدول زیر عملگرهای ریاضی Go را نشان می‌دهد:

عملگر	مثال	نتیجه
+	var1 = var2 + var3	Var1 برابر است با حاصل جمع var2 و var3
-	var1 = var2 - var3	Var1 برابر است با حاصل تفریق var2 و var3
*	var1 = var2 * var3	Var1 برابر است با حاصلضرب var2 در var3
/	var1 = var2 / var3	Var1 برابر است با حاصل تقسیم var2 بر var3
%	var1 = var2 % var3	Var1 برابر است با باقیمانده تقسیم var2 و var3
++	var1 ++	به متغیر var1 یک واحد اضافه می کند.

--	var1 --	از متغیر var1 یک واحد کم می کند.

مثال بالا در از نوع عددی استفاده شده است. اما استفاده از عملگرهای ریاضی برای نوع رشته‌ای نتیجه متفاوتی دارد. اگر از عملگر + برای رشته‌ها استفاده کنیم دو رشته را با هم ترکیب کرده و به هم می‌چسباند. حال می‌توانیم با ایجاد یک برنامه نحوه عملکرد عملگرهای ریاضی در Go را یاد بگیریم:

```go
package main

import "fmt"

func main() {

    //Assign test values
    var num1 int = 5
    var num2 int = 3

    //Demonstrate use of mathematical operators
    fmt.Printf("The sum of %d and %d is %d.              \n", num1, num2, (num1 + num2))
    fmt.Printf("The difference of %d and %d is %d.       \n", num1, num2, (num1 - num2))
    fmt.Printf("The product of %d and %d is %d.          \n", num1, num2, (num1 * num2))
    fmt.Printf("The quotient of %d and %d is %.2f.       \n", num1, num2, float32(num1 / num2))
    fmt.Printf("The remainder of %d divided by %d is %d.\n", num1, num2, (num1 % num2))
    fmt.Printf("The result of %d power %d is %d.         \n", num1, num2, (num1 * num2))
    fmt.Printf("The quotient of %d and %d is %d.         \n", num1, num2, (num1 / num2))

    //Demonstrate concatenation on strings using the + operator
    var msg1 = "Hello "
    var msg2 = "World!"
    fmt.Print(msg1 + msg2)
}
```

```
The sum of 5 and 3 is 8.
The difference of 5 and 3 is 2.
The product of 5 and 3 is 15.
The quotient of 5 and 3 is 1.00.
The remainder of 5 divided by 3 is 2.
The result of 5 power 3 is 125.
The quotient of 5 and 3 is 1.
Hello World!
```

برنامه بالا نتیجه هر عبارت را نشان می‌دهد. در این برنامه از متد Printf() برای نشان دادن نتایج و از \n برای چاپ آنها در سطرهای متفاوت استفاده شده است. در خط ۱۵ برای اینکه ارقام کسری بعد از عدد حاصل دو رقم باشند از %.2f استفاده می‌کنیم %.2f. در این جا بدین معناست که عدد را تا دو رقم اعشار نمایش بده. Go خط جدید و فاصله و فضای خالی را نادیده می‌گیرد. در خط ۲۳ مشاهده می‌کنید که دو رشته به وسیله عملگر + به هم متصل شده‌اند. نتیجه استفاده از عملگر + برای چسباندن دو کلمه "Hello " و "World!" رشته "Hello World!" خواهد بود. به فاصله‌های خالی بعد از اولین کلمه توجه کنید اگر آنها را حذف کنید از خروجی برنامه نیز حذف می‌شوند.

عملگرهای مقایسه ای

از عملگرهای مقایسه‌ای برای مقایسه مقادیر استفاده می‌شود. نتیجه این مقادیر یک مقدار بولی (منطقی) است. این عملگرها اگر نتیجه مقایسه دو مقدار درست باشد مقدار ۱ و اگر نتیجه مقایسه اشتباه باشد مقدار ۰ را نشان می‌دهند. این عملگرها به طور معمول در دستورات شرطی به کار می‌روند به این ترتیب که باعث ادامه یا توقف دستور شرطی می‌شوند. جدول زیر عملگرهای مقایسه‌ای در Go را نشان می‌دهد:

عملگر	مثال	نتیجه
==	var1 = var2 == var3	var1 در صورتی True است که مقدار var2 با مقدار var3 برابر باشد در غیر اینصورت False است
!=	var1 = var2 != var3	var1 در صورتی True است که مقدار var2 با مقدار var3 برابر نباشد در غیر اینصورت False است
<	var1 = var2 < var3	var1 در صورتی True است که مقدار var2 کوچک‌تر از var3 مقدار باشد در غیر اینصورت False است
>	var1 = var2 > var3	var1 در صورتی True است که مقدار var2 بزرگ‌تر ازمقدار var3 باشد در غیر اینصورت False است
<=	var1 = var2 <= var3	var1 در صورتی True است که مقدار var2 کوچک‌تر یا مساوی مقدار var3 باشد در غیر اینصورت False است
>=	var1 = var2 >= var3	var1 در صورتی True است که مقدار var2 بزرگ‌تر یا مساوی var3 مقدار باشد در غیر اینصورت False است

برنامه زیر نحوه عملکرد این عملگرها را نشان می‌دهد:

```go
package main

import "fmt"

func main() {
    var num1 int = 10;
    var num2 int = 5;

    fmt.Printf("%d == %d : %t \n", num1, num2, num1 == num2)
    fmt.Printf("%d != %d : %t \n", num1, num2, num1 != num2)
    fmt.Printf("%d <  %d : %t \n", num1, num2, num1  < num2)
    fmt.Printf("%d >  %d : %t \n", num1, num2, num1  > num2)
    fmt.Printf("%d <= %d : %t \n", num1, num2, num1 <= num2)
    fmt.Printf("%d >= %d : %t \n", num1, num2, num1 >= num2)
}
```

```
10 == 5 : False
10 != 5 : True
10 <> 5 : True
10 <  5 : False
10 >  5 : True
10 <= 5 : False
 : ۵ =< ۱۰True
```

در مثال بالا ابتدا دو متغیر را که می‌خواهیم با هم مقایسه کنیم را ایجاد کرده و به آنها مقادیری اختصاص می‌دهیم. سپس با استفاده از یک عملگر مقایسه‌ای آنها را با هم مقایسه کرده و نتیجه را چاپ می‌کنیم. به این نکته توجه کنید که هنگام مقایسه دو متغیر از عملگر == به جای عملگر = باید استفاده شود. عملگر = عملگر تخصیصی است و در عبارتی مانند x = y مقدار y را در به x اختصاص می‌دهد. عملگر == عملگر مقایسه‌ای است که دو مقدار را با هم مقایسه می‌کند مانند x==y و اینطور خوانده می‌شود x برابر است با y .

عملگرهای منطقی

عملگرهای منطقی بر روی عبارات منطقی عمل می‌کنند و نتیجه آنها نیز یک مقدار بولی است. از این عملگرها اغلب برای شرطهای پیچیده استفاده می‌شود. همانطور که قبلاً یاد گرفتید، مقادیر بولی می‌توانند `false` یا `true` باشند. فرض کنید که `var2` و `var3` دو مقدار بولی هستند:

عملگر	نام	مثال
&&	منطقی AND	var1 = var2 && var3
||	منطقی OR	var1 = var2 || var3
!	منطقی NOT	var1 = !var1

عملگر منطقی (&&)AND

اگر مقادیر دو طرف عملگر AND، `true` باشند عملگر AND مقدار `true` را بر می‌گرداند. در غیر اینصورت اگر یکی از مقادیر یا هر دوی آنها `false` باشند مقدار `false` را بر می‌گرداند. در زیر جدول درستی عملگر AND نشان داده شده است :

X && Y	Y	X
true	true	true

false	false	true
false	true	false
false	false	false

برای درک بهتر تأثیر عملگر AND یاد آوری می‌کنم که این عملگر فقط در صورتی مقدار true را نشان می‌دهد که هر دو عملوند مقدارشان true باشد. در غیر اینصورت نتیجه تمام ترکیبهای بعدی false خواهد شد. استفاده از عملگر AND مانند استفاده از عملگرهای مقایسه‌ای است. به عنوان مثال نتیجه عبارت زیر درست (true) است اگر سن (age) بزرگ‌تر از ۱۸ و salary کوچک‌تر از ۱۰۰۰ باشد.

```
var result bool = (age > 18) && (salary < 1000)
```

عملگر AND زمانی کارامد است که ما با محدود خاصی از اعداد سرو کار داریم. مثلاً عبارت 10 <= x <= 100 بدین معنی است که x می‌تواند مقداری شامل اعداد ۱۰ تا ۱۰۰ را بگیرد. حال برای انتخاب اعداد خارج از این محدوده می‌توان از عملگر منطقی AND به صورت زیر استفاده کرد.

```
var inRange bool = (number <= 10) && (number >= 100)
```

عملگر منطقی OR(||)

اگر یکی یا هر دو مقدار دو طرف عملگر OR، درست (true) باشد، عملگر OR مقدار true را بر می‌گرداند. جدول درستی عملگر OR در زیر نشان داده شده است :

X	Y	X || Y
true	true	true
true	false	true
false	true	true
false	false	false

در جدول بالا مشاهده می‌کنید که عملگر OR در صورتی مقدار false را بر می‌گرداند که مقادیر دو طرف آن false باشند. کد زیر را در نظر بگیرید. نتیجه این کد در صورتی درست (true) است که رتبه نهایی دانش آموز (finalGrade) بزرگ‌تر از ۷۵ یا نمره نهایی امتحان آن ۱۰۰ باشد.

```
var isPassed bool = (finalGrade >= 75) || (finalExam == 100)
```

عملگر منطقی (!)NOT

برخلاف دو اپراتور OR و AND عملگر منطقی NOT یک عملگر یگانی است و فقط به یک عملوند نیاز دارد. این عملگر یک مقدار یا اصطلاح بولی را نفی می‌کند. مثلاً اگر عبارت یا مقدار true باشد آنرا false و اگر false باشد آنرا true می‌کند. جدول زیر عملکرد اپراتور NOT را نشان می‌دهد :

X	!X
true	false
false	true

نتیجه کد زیر در صورتی درست است که age (سن) بزرگ‌تر یا مساوی ۱۸ نباشد.

```
var isMinor bool = !(age >= 18)
```

عملگرهای بیتی

عملگرهای بیتی به شما اجازه می‌دهند که شکل باینری انواع داده‌ها را دستکاری کنید. برای درک بهتر این درس توصیه می‌شود که شما سیستم باینری و نحوه تبدیل اعداد دهدهی به باینری را یاد بگیرید. در سیستم باینری (دودویی) که کامپیوتر از آن استفاده می‌کند وضعیت هر چیز یا خاموش است یا روشن. برای نشان دادن حالت روشن از عدد ۱ و برای نشان دادن حالت خاموش از عدد ۰ استفاده می‌شود. بنابراین اعداد باینری فقط می‌توانند صفر یا یک باشند. اعداد باینری را اعداد در مبنای ۲ و اعداد اعشاری را اعداد در مبنای ۱۰ می گویند. یک بیت نشان دهنده یک رقم باینری است و هر بایت نشان دهنده ۸ بیت است. به عنوان مثال برای یک داده از نوع int به ۳۲ بیت یا ۸ بایت فضا برای ذخیره آن نیاز داریم، این بدین معناست که اعداد از ۳۲ رقم ۰ و ۱ برای ذخیره استفاده می‌کنند. برای مثال عدد ۱۰۰ وقتی به عنوان یک متغیر از نوع int ذخیره می‌شود در کامپیوتر به صورت زیر خوانده می‌شود :

```
00000000000000000000000001100100
```

عدد ۱۰۰ در مبنای ده معادل عدد ۱۱۰۰۱۰۰ در مبنای ۲ است. در اینجا ۷ رقم سمت راست نشان دهنده عدد ۱۰۰ در مبنای ۲ است و مابقی صفرهای سمت راست برای پر کردن بیتهایی است که عدد از نوع int نیاز دارد. به این نکته توجه کنید که اعداد باینری از سمت راست به چپ خوانده می‌شوند. عملگرهای بیتی Go در جدول زیر نشان داده شده‌اند :

عملگر	نام	مثال
&	بیتی AND	x = y & z
|	بیتی OR	x = y | z
^	بیتی XOR	x = y ^ z
&=	بیتی تخصیصیAND	x &= y
|=	بیتی تخصیصی OR	x |= y
^=	بیتی تخصیصیXOR	x ^= y

عملگر بیتی AND(&)

عملگر بیتی AND کاری شبیه عملگر منطقی AND انجام می‌دهد با این تفاوت که این عملگر بر روی بیتها کار می‌کند. اگر مقادیر دو طرف آن ۱ باشد مقدار ۱ را بر می‌گرداند و اگر یکی یا هر دو طرف آن صفر باشد مقدار صفر را بر می‌گرداند. جدول درستی عمگر بیتی AND در زیر آمده است:

X AND Y	Y	X
1	1	1
0	0	1
0	1	0
0	0	0

در زیر نحوه استفاده از عملگر بیتی AND آمده است :

```
var result int = 5 & 3

fmt.Println(result)
```

```
1
```

همانطور که در مثال بالا مشاهده می‌کنید نتیجه عملکرد عملگر AND بر روی دو مقدار ۵ و ۳ عدد یک می‌شود. اجازه بدهید ببینیم که چطور این نتیجه را به دست می‌آید:

```
5:  00000000000000000000000000000101
3:  00000000000000000000000000000011
----------------------------------
1:  00000000000000000000000000000001
```

ابتدا دو عدد ۵ و ۳ به معادل باینری‌شان تبدیل می‌شوند. از آنجاییکه هر عدد صحیح (int) ۳۲ بیت است از صفر برای پر کردن بیتهای خالی استفاده می‌کنیم. با استفاده از جدول درستی عملگر بیتی AND می‌توان فهمید که چرا نتیجه عدد یک می‌شود.

عملگر بیتی OR(|)

اگر مقادیر دو طرف عملگر بیتی OR هر دو صفر باشند، نتیجه صفر، در غیر اینصورت ۱ خواهد شد. جدول درستی این عملگر در زیر آمده است :

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

نتیجه عملگر بیتی OR در صورتی صفر است که عملوندهای دو طرف آن صفر باشند. اگر فقط یکی از دو عملوند یک باشد نتیجه یک خواهد شد. به مثال زیر توجه کنید :

```
var result int= 7 | 9;

fmt.Println(result)
```

```
15
```

وقتی که از عملگر بیتی OR برای دو مقدار در مثال بالا (۷ و ۹) استفاده می‌کنیم نتیجه ۱۵ می‌شود. حال بررسی می‌کنیم که چرا این نتیجه به دست آمده است؟

```
 7: 00000000000000000000000000000111
 9: 00000000000000000000000000001001
---------------------------------
15: 00000000000000000000000000001111
```

با استفاده از جدول درستی عملگر بیتی OR می‌توان نتیجه استفاده از این عملگر را تشخیص داد. عدد ۱۱۱۱ باینری معادل عدد ۱۵ صحیح است.

عملگر بیتی (^)XOR

جدول درستی این عملگر در زیر آمده است :

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

در صورتیکه عملوندهای دو طرف این عملگر هر دو صفر یا هر دو یک باشند نتیجه صفر در غیر اینصورت نتیجه یک می‌شود. در مثال زیر تأثیر عملگر بیتی XOR را بر روی دو مقدار مشاهده می‌کنید :

```
var result int = 5 ^ 7

fmt.Println(result)
```

```
2
```

در زیر معادل باینری اعداد بالا (۵ و ۷) نشان داده شده است.

```
5: 00000000000000000000000000000101
7: 00000000000000000000000000000111
-----------------------------------
2: 00000000000000000000000000000010
```

با نگاه کردن به جدول درستی عملگر بیتی XOR ، می‌توان فهمید که چرا نتیجه عدد ۲ می‌شود.

عملگر بیتی تغییر مکان (shift)

این نوع عملگرها به شما اجازه می‌دهند که بیتها را به سمت چپ یا راست جا به جا کنید. دو نوع عملگر بیتی تغییر مکان وجود دارد که هر کدام دو عملوند قبول می‌کنند. عملوند سمت چپ این عملگرها حالت باینری یک مقدار و عملوند سمت راست تعداد جابه جایی بیت ها را نشان می‌دهد.

عملگر	نام	مثال
>>	تغییر مکان به سمت چپ	x = y << 2
<<	تغییر مکان به سمت راست	x = y >> 2

عملگر تغییر مکان به سمت چپ

این عملگر، بیتهای عملوند سمت چپ را به تعداد n مکان مشخص شده، توسط عملوند سمت راست، به سمت چپ منتقل می‌کند. به عنوان مثال :

```
var result int = 10 << 2

fmt.Println(result)
```

```
40
```

در مثال بالا، ما بیتهای مقدار ۱۰ را دو مکان به سمت چپ منتقل کرده‌ایم، حال بیایید تأثیر این انتقال را بررسی کنیم :

```
10: 00000000000000000000000000001010
------------------------------------
40: 00000000000000000000000000101000
```

مشاهده می‌کنید که همه بیت‌ها به اندازه دو واحد به سمت چپ منتقل شده‌اند. در این انتقال دو صفر از صفرهای سمت چپ کم می‌شود و در عوض دو صفر به سمت راست اضافه می‌شود.

عملگر تغییر مکان به سمت راست

این عملگر شبیه به عمگر تغییر مکان به سمت چپ است با این تفاوت که بیت‌ها را به سمت راست جا به جا می‌کند. به عنوان مثال :

```
var result int = 100 >> 4

fmt.Println(result)
```

```
6
```

با استفاده از عملگرتغییر مکان به سمت راست بیت‌های مقدار ۱۰۰ را به اندازه ۴ واحد به سمت چپ جا به جا می‌کنیم. اجازه بدهید تأثیر این جا به جایی را مورد بررسی قرار دهیم :

```
100: 00000000000000000000000001100100
------------------------------------
```

```
6: 00000000000000000000000000000110
```

هر بیت به اندازه ۴ واحد به سمت راست منتقل می‌شود، بنابراین ۴ بیت اول سمت راست حذف شده و چهار صفر به سمت چپ اضافه می‌شود.

عملگرهای تخصیصی

نوع دیگر از عملگرهای Go عملگرهای جایگزینی نام دارند.این عملگرها مقدار متغیر سمت راست خود را در متغیر سمت چپ قرار می دهند.

جدول زیر انواع عملگرهای تخصیصی در Go را نشان می دهد:

عملگر	مثال	نتیجه
=	var1 = var2	مقدار var1 برابر است با مقدار var2
+=	var1 += var2	مقدار var1 برابر است با حاصل جمع var1 وvar2
-=	var1 -= var2	مقدار var1 برابر است با حاصل تفریق var1 وvar2
*=	var1 *= var2	مقدار var1 برابر است با حاصل ضرب var1 درvar2
/=	var1 /= var2	مقدار var1 برابر است با حاصل تقسیم var1 برvar2
%=	var1 %= var2	مقدار var1 برابر است با باقیمانده تقسیم var1 برvar2

از عملگر =+ برای اتصال دو رشته نیز می توان استفاده کرد.استفاده از این نوع عملگرها در واقع یک نوع خلاصه نویسی در کد است. مثلا شکل اصلی کد var1 += var2 به صورت var1 = var1 + var2 می باشد. این حالت کدنویسی زمانی کارایی خود را نشان می دهد که نام متغیرها طولانی باشد. برنامه زیر چگونگی استفاده از عملگرهای تخصیصی و تاثیر آنها را بر متغیر ها نشان می دهد.

```go
package main

import "fmt"

func main() {
    var number int

    fmt.Println("Assigning 10 to number...")
    number = 10
    fmt.Printf("Number = %d \n", number)

    fmt.Println("Adding 10 to number...")
    number += 10
```

```
    fmt.Printf("Number = %d \n", number)

    fmt.Println("Subtracting 10 to number...")
    number -= 10
    fmt.Printf("Number = %d", number)
}
```

```
Assigning 10 to number...
Number = 10
Adding 10 to number...
Number = 20
Subtracting 10 from number...
Number = 10
```

در برنامه از ۳ عملگر تخصیصی استفاده شده است. ابتدا یک متغیر و مقدار 10 با استفاده از عملگر = به آن اختصاص داده شده است.سپس به آن با استفاده از عملگر =+ مقدار ۱۰ اضافه شده است.و در آخر به وسیله عملگر =- عدد ۱۰ از آن کم شده است.

عملگرهای متفرقه

علاوه بر عملگرهایی که تا اینجا با آنها آشنا شدید، Go دارای عملگرهای دیگری هم هست که در جدول زیر به آنها اشاره شده است:

عملگر	توضیح
&	آدرس یک متغیر را بر می گرداند.
*	به یک متغیر اشاره می کند.

کاربرد این عملگرها را در درس های آینده، به شما آموزش می دهیم.

تقدم عملگرها

تقدم عملگرها، مشخص می‌کند که در محاسباتی که بیش از دو عملوند دارند، ابتدا کدام عملگر اثرش را اعمال کند. عملگرها در Go در محاسبات دارای حق تقدم هستند. به عنوان مثال :

```
var number int = 1 + 2 * 3 / 1
```

اگر ما حق تقدم عملگرها را رعایت نکنیم و عبارت بالا را از سمت چپ به راست انجام دهیم نتیجه ۹ خواهد شد (۳=۲+۱ سپس ۹=۳×۳ و در آخر ۹=۹/۱). اما کامپایلر با توجه به تقدم عملگرها محاسبات را انجام می‌دهد. برای مثال عمل ضرب و تقسیم نسبت به جمع و تفریق

تقدم دارند. بنابراین در مثال فوق ابتدا عدد ۲ ضربدر ۳ و سپس نتیجه آنها تقسیم بر ۱ می‌شود که نتیجه ۶ به دست می‌آید. در آخر عدد ۶ با ۱ جمع می‌شود و عدد ۷ حاصل می‌شود. در جدول زیر تقدم برخی از عملگرهای Go آمده است :

تقدم	عملگر
بالاترین	- - ++ . -> [] ()
	+ - ! ~ ++ - - (type)* & sizeof
	* / %
	+ -
	<< >>
	< <= > >=
	== !=
	&
	^
	|
	&&
	||
	?:
	= |= &= ^= <<= >>= %= /= *= -= +=
پایین ترین	,

ابتدا عملگرهای با بالاترین و سپس عملگرهای با پایین‌ترین حق تقدم در محاسبات تأثیر می‌گذارند. برای ایجاد خوانایی در تقدم عملگرها و انجام محاسباتی که در آنها از عملگرهای زیادی استفاده می‌شود از پرانتز استفاده می‌کنیم :

```
var number int = ( 1 + 2 ) * ( 3 / 4 ) % ( 5 - ( 6 * 7 ))
```

در مثال بالا ابتدا هر کدام از عباراتی که داخل پرانتز هستند مورد محاسبه قرار می‌گیرند. به نکته‌ای در مورد عبارتی که در داخل پرانتز سوم قرار دارد توجه کنید. در این عبارت ابتدا مقدار داخلی‌ترین پرانتز مورد محاسبه قرار می‌گیرد یعنی مقدار ۶ ضربدر ۷ شده و سپس از ۵ کم می‌شود. اگر دو یا چند عملگر با حق تقدم یکسان موجود باشد ابتدا باید هر کدام از عملگرها را که در ابتدای عبارت می‌آیند مورد ارزیابی قرار دهید. به عنوان مثال :

```
var number int = 3 * 2 + 8 / 4
```

هر دو عملگر * و / دارای حق تقدم یکسانی هستند. بنابر این شما باید از چپ به راست آنها را در محاسبات تأثیر دهید. یعنی ابتدا ۳ را ضربدر ۲ می‌کنید و سپس عدد ۸ را بر ۴ تقسیم می‌کنید. در نهایت نتیجه دو عبارت را جمع کرده و در متغیر number قرار می‌دهید.

گرفتن ورودی از کاربر

Go متد Scanln() را برای گرفتن ورودی از کاربر، در اختیار شما قرار می‌دهد. این متد، تمام کاراکترهایی را که شما در محیط برنامه نویسی تایپ می‌کنید تا زمانی که دکمه Enter را می‌زنید، می‌خواند. به برنامه زیر توجه کنید :

```
package main

import "fmt"

func main() {
    var name   string
    var age    int
    var height float32

    fmt.Print("Enter your name: ")
    fmt.Scanln(&name);

    fmt.Print("Enter your age: ")
    fmt.Scanln(&age);

    fmt.Print("Enter your height: ")
    fmt.Scanln(&height);

    //Print a blank line
    fmt.Println();

    //Show the details you typed
    fmt.Printf("Name is %s.     \n", name)
    fmt.Printf("Age is %d.      \n", age)
    fmt.Printf("Height is %.1f. \n", height)
}
```

```
Enter your name: John
Enter your age: 18
Enter your height: 160.5

Name is John.
Age is 18.
Height is 160.5.
```

ابتدا ۳ متغیر را برای ذخیره داده در برنامه تعریف می‌کنیم (خطوط ۶ و ۷ و ۸). برنامه از کاربر می‌خواهد که نام خود را وارد کند (خط ۱۰). در خط ۱۱ شما به عنوان کاربر نام خود را وارد می‌کنید. سپس برنامه از ما سن را سؤال می‌کند (خط ۱۳) و در نهایت در خطوط ۱۶ و ۱۷، برنامه از ما قدمان (height) را می خواهد. در خط ۲۰ هم یک خط فاصله به وسیله متد ()Println ایجاد کرده ایم تا بین ورودی های شما و خروجی فاصله ای جهت تفکیک ایجاد شود. حال برنامه را اجرا کرده و با وارد کردن مقادیر مورد نظر نتیجه را مشاهده کنید. یک نکته که نباید فراموش شود، وجود علامت & قبل از نام متغیرها و در داخل متد ()Scanln است (خطوط ۱۱، ۱۴ و ۱۷). این وجود این عملگر در داخل متد ()Scanln و قبل از نام متغیرها به برنامه این امکان را می دهد، که منتظر بماند تا شما مقدار متغیر مربوطه را از طریق صفحه کلید و در هنگام اجرای برنامه وارد کنید.

ساختارهای تصمیم

تقریباً همه زبانهای برنامه نویسی به شما اجازه اجرای کد را در شرایط مطمئن می‌دهند. حال تصور کنید که یک برنامه دارای ساختار تصمیم گیری نباشد و همه کدها را اجرا کند. این حالت شاید فقط برای چاپ یک پیغام در صفحه مناسب باشد ولی فرض کنید که شما بخواهید اگر مقدار یک متغیر با یک عدد برابر باشد سپس یک پیغام چاپ شود آن وقت با مشکل مواجه خواهید شدGo . راه‌های مختلفی برای رفع این نوع مشکلات ارائه می‌دهد. در این بخش با مطالب زیر آشنا خواهید شد :

- **دستور**if
- **دستور**if…else
- **دستور** if **چندگانه**
- **دستور** if **تو در تو**
- **دستور**switch

دستور if

می‌توان با استفاده از دستور if و یک شرط خاص که باعث ایجاد یک کد می‌شود یک منطق به برنامه خود اضافه کنید. دستور if ساده‌ترین دستور شرطی است که برنامه می‌گوید اگر شرطی برقرار است، کد معینی را انجام بده. ساختار دستور if به صورت زیر است :

```
if condition {
   code to execute
}
```

قبل از اجرای دستور if ابتدا شرط بررسی می‌شود. اگر شرط برقرار باشد یعنی درست باشد سپس کد اجرا می‌شود. شرط یک عبارت مقایسه‌ای است. می‌توان از عملگرهای مقایسه‌ای برای تست درست یا اشتباه بودن شرط استفاده کرد. اجازه بدهید که نگاهی به نحوه استفاده از دستور if در داخل برنامه بیندازیم. برنامه زیر پیغام Hello World را اگر مقدار number کمتر از ۱۰ و Goodbye World را اگر مقدار number از ۱۰ بزرگ‌تر باشد در صفحه نمایش می‌دهد.

```
1  package main
2
3  import "fmt"
4
5  func main() {
6      //Declare a variable and set it a value less than 10
7      var number int = 5
8
9      //If the value of number is less than 10
10     if number < 10 {
11         fmt.Println("Hello World.")
12     }
13
14     //Change the value of a number to a value which is greater than 10
15     number = 15
16
17     //If the value of number is greater than 10
18     if number > 10 {
19         fmt.Println("Goodbye World.")
20     }
21 }
```

```
Hello World.
Goodbye World.
```

در خط ۷ یک متغیر با نام number تعریف و مقدار ۵ به آن اختصاص داده شده است. وقتی به اولین دستور if در خط ۱۰ می‌رسیم برنامه تشخیص می‌دهد که مقدار number از ۱۰ کمتر است یعنی ۵ کوچک‌تر از ۱۰ است.

منطقی است که نتیجه مقایسه درست می‌باشد. بنابراین دستور if دستور را اجرا می‌کند (خط ۱۱) و پیغام Hello World چاپ می‌شود. حال مقدار number را به ۱۵ تغییر می‌دهیم (خط ۱۵). وقتی به دومین دستور if در خط ۱۸ می‌رسیم برنامه مقدار number را با ۱۰ مقایسه می‌کند و چون مقدار number یعنی ۱۵ از ۱۰ بزرگ‌تر است برنامه پیغام Goodbye World را چاپ می‌کند (خط ۱۹). شما می‌توانید چندین دستور را در داخل دستور if بنویسید. کافیست که از یک آکولاد برای نشان دادن ابتدا و انتهای دستورات استفاده کنید. همه دستورات داخل بین آکولاد جز بدنه دستور if هستند. نحوه تعریف چند دستور در داخل بدنه if به صورت زیر است :

```
if condition {
    statement1
    statement2
```

```
    .
    .
    .
    statementN
}
```

این هم یک مثال ساده :

```
if x > 10 {
    fmt.Println("x is greater than 10.")
    fmt.Println("This is still part of the if statement.")
}
```

در مثال بالا اگر مقدار x از ۱۰ بزرگ‌تر باشد دو پیغام چاپ می‌شود.

دستور if else

دستور if فقط برای اجرای یک حالت خاص به کار می‌رود یعنی اگر حالتی برقرار بود کار خاصی انجام شود. اما زمانی که شما بخواهید اگر شرط خاصی برقرار شد یک دستور و اگر برقرار نبود دستور دیگر اجرا شود باید از دستور if else استفاده کنید. ساختار دستور if else در زیر آمده است :

```
if  condition {
   code to execute if condition is true
} else {
   code to execute if condition is false
}
```

از کلمه کلیدی else نمی‌توان به تنهایی استفاده کرد بلکه حتماً باید با if به کار برده شود. اگر فقط یک کد اجرایی در داخل بدنه if و بدنه else دارید استفاده از آکولاد اختیاری است. کد داخل بلوک else فقط در صورتی اجرا می‌شود که شرط داخل دستور if نادرست باشد. در زیر نحوه استفاده از دستور if…else آمده است.

```
1  package main
2
3  func main() {
4      var number int = 5
5
6      //Test the condition
7      if number < 10 {
8          println("The number is less than 10.")
9      } else {
10         print("The number is either greater than or equal to 10.")
11     }
```

```
12
13      //Modify value of number
14      number = 15
15
16      //Repeat the test to yield a different result
17      if number < 10 {
18          println("The number is less than 10.")
19      } else {
20          println("The number is either greater than or equal to 10.")
21      }
22  }
```

```
The number is less than 10.
The number is either greater than or equal to 10.
```

در خط ۴ یک متغیر به نام number تعریف کرده‌ایم و در خط ۷ تست می‌کنیم که آیا مقدار متغیر number از ۱۰ کمتر است یا نه و چون کمتر است، در نتیجه، کد داخل بلوک if اجرا می‌شود (خط ۸) و اگر مقدار number را تغییر دهیم و به مقداری بزرگتر از ۱۰ تغییر دهیم (خط ۱۴)، شرط نادرست می‌شود (خط ۱۷) و کد داخل بلوک else اجرا می‌شود (خط ۱۸).

دستور if…else if…else

اگر بخواهید چند شرط را بررسی کنید چکار می‌کنید؟ می‌توانید از چندین دستور if استفاده کنید و بهتر است که این دستورات if را به صورت زیر بنویسید :

```
if condition {
    code to execute
} else {
    if condition {
        code to execute
    }
} else {
    if condition {
        code to execute
    }
} else {
    code to execute
}
```

خواندن کد بالا سخت است. بهتر است دستورات را به صورت تو رفتگی در داخل بلوک else بنویسید. می‌توانید کد بالا را ساده‌تر کنید :

```
if condition {
    code to execute
} else if condition {
    code to execute
} else if condition {
```

```
code to execute
        } else {
code to execute
        }
```

حال که نحوه استفاده از دستور if else را یاد گرفتید باید بدانید که مانند else ، else if نیز به دستور if وابسته است. دستور else if وقتی اجرا می‌شود که اولین دستور if اشتباه باشد حال اگر else if اشتباه باشد دستور else if بعدی اجرا می‌شود. و اگر آن نیز اجرا نشود در نهایت دستور else اجرا می‌شود. برنامه زیر نحوه استفاده از دستورelse if را نشان می‌دهد :

```go
package main

import "fmt"

func main() {
    fmt.Println("What's your favorite color?")
    fmt.Println("[1] Black")
    fmt.Println("[2] White")
    fmt.Println("[3] Blue")
    fmt.Println("[4] Red")
    fmt.Println("[5] Yellown")

    var choice int

    fmt.Print("Enter your choice: ")
    fmt.Scanln(&choice)

    if (choice == 1) {
        print("You might like my black t-shirt.")
    } else if (choice == 2) {
        print("You might be a clean and tidy person.")
    } else if (choice == 3) {
        print("You might be sad today.")
    } else if (choice == 4) {
        print("You might be inlove right now.")
    } else if (choice == 5) {
        print("Lemon might be your favorite fruit.")
    } else {
        print("Sorry, your favorite color is not in the choices above.")
    }
}
```

```
What's your favorite color?
[1] Black
[2] White
[3] Blue
[4] Red
[5] Yellow

Enter your choice: 1
You might like my black t-shirt.
What's your favorite color?
[1] Black
```

```
[2] White
[3] Blue
[4] Red
[5] Yellow

Enter your choice: 999
Sorry, your favorite color is not in the choices above.
```

خروجی برنامه بالا به متغیر choice وابسته است. بسته به اینکه شما چه چیزی انتخاب می‌کنید پیغامهای مختلفی چاپ می‌شود. اگر عددی که شما تایپ می‌کنید در داخل حالتهای انتخاب نباشد کد مربوط به بلوک else اجرا می‌شود.

دستور if تو در تو

می‌توان از دستور if تو در تو در Go استفاده کرد. یک دستور ساده if در داخل دستور if دیگر.

```
if condition {
    code to execute

    if  condition {
         code to execute
    }
    else if condition {
        if condition {
           code to execute
        }
    }
}
else {
    if condition {
       code to execute
    }
}
```

اجازه بدهید که نحوه استفاده از دستور if تو در تو را نشان دهیم :

```
1  package main
2
3  import "fmt"
4
5  func main() {
6
7      var age    int
8      var gender string
9
10     fmt.Print("Enter your age: ")
11     fmt.Scanln(&age)
```

```
12
13      fmt.Print("Enter your gender: ")
14      fmt.Scanln(&gender)
15
16      if (age > 12) {
17          if (age < 20) {
18              if (gender == "male") {
19                  print("You are a teenage boy.")
20              } else {
21                  print("You are a teenage girl.")
22              }
23          } else {
24              print("You are already an adult.")
25          }
26      } else {
27          print("You are still too young.")
28      }
29  }
```

```
Enter your age: 18
Enter your gender: male
You are a teenage boy.
Enter your age: 12
Enter your gender: female
You are still too young.
```

اجازه بدهید که برنامه را کالبد شکافی کنیم. ابتدا برنامه از شما درباره سنتان سؤال می‌کند (خط ۱۰). در خط ۱۳ درباره جنستان از شما سؤال می‌کند. سپس به اولین دستور `if` می‌رسد (خط ۱۶). در این قسمت اگر سن شما بیشتر از 12 سال باشد برنامه وارد بدنه دستور `if` می‌شود در غیر اینصورت وارد بلوک `else` (خط ۲۶) مربوط به همین دستور `if` می‌شود.

حال فرض کنیم که سن شما بیشتر از ۱۲ سال است و شما وارد بدنه اولین `if` شده‌اید. در بدنه اولین `if` دو دستور `if` دیگر را مشاهده می‌کنید. اگر سن کمتر ۲۰ باشد شما وارد بدنه `if` دوم می‌شوید و اگر نباشد به قسمت `else` متناظر با آن می‌روید (خط ۲۳). دوباره فرض می‌کنیم که سن شما کمتر از ۲۰ باشد، در اینصورت وارد بدنه `if` دوم شده و با یک `if` دیگر مواجه می‌شوید (خط ۱۸). در اینجا جنسیت شما مورد بررسی قرار می‌گیرد که اگر برابر "male" باشد، کدهای داخل بدنه سومین `if` اجرا می‌شود در غیر اینصورت قسمت `else` مربوط به این `if` اجرا می‌شود (خط ۲۰). پیشنهاد می‌شود که از `if` تو در تو در برنامه کمتر استفاده کنید چون خوانایی برنامه را پایین می‌آورد.

دستور switch

در Go ساختاری به نام switch وجود دارد که به شما اجازه می‌دهد که با توجه به مقدار ثابت یک متغیر چندین انتخاب داشته باشید. دستور switch معادل دستور if…else if است با این تفاوت که در دستور switch متغیر فقط مقادیر ثابتی از اعداد، رشته‌ها و یا کاراکترها را قبول می‌کند. مقادیر ثابت مقادیری هستند که قابل تغییر نیستند. در زیر نحوه استفاده از دستور switch آمده است :

```
switch (testVar)
{
   case compareVal1:
       code to execute if testVar == compareVa11
   case compareVa12:
       code to execute if testVar == compareVa12
   .
   .
   .
   case compareVa1N:
       code to execute if testVer == compareVa1N
   default:
       code to execute if none of the values above match the testVar
}
```

ابتدا یک مقدار در متغیر switch که در مثال بالا testVar است قرار می‌دهید. این مقدار با هر یک از عبارتهای case داخل بلوک switch مقایسه می‌شود. اگر مقدار متغیر با هر یک از مقادیر موجود در دستورات case برابر بود کد مربوط به آن case اجرا خواهد شد. به این نکته توجه کنید که حتی اگر تعداد خط کدهای داخل دستور case از یکی بیشتر باشد نباید از آکولاد استفاده کنیم. دستور switch یک بخش default دارد. این دستور در صورتی اجرا می‌شود که مقدار متغیر با هیچ یک از مقادیر دستورات case برابر نباشد. دستور default اختیاری است و اگر از بدنه switch حذف شود هیچ اتفاقی نمی‌افتد. مکان این دستور هم مهم نیست اما بر طبق تعریف آن را در پایان دستورات می‌نویسند. به مثالی در مورد دستور switch توجه کنید :

```
1  package main
2
3  import "fmt"
4
5  func main() {
6      var choice int
7
8      fmt.Println("What's your favorite pet?")
9      fmt.Println("[1] Dog")
10     fmt.Println("[2] Cat")
11     fmt.Println("[3] Rabbit")
12     fmt.Println("[4] Turtle")
13     fmt.Println("[5] Fish")
14     fmt.Println("[6] Not in the choices\n")
15     fmt.Print("Enter your choice: ")
```

```
16
17      fmt.Scanln(&choice)
18
19      switch (choice) {
20          case 1:
21              fmt.Println("Your favorite pet is Dog.")
22          case 2:
23              fmt.Println("Your favorite pet is Cat.")
24          case 3:
25              fmt.Println("Your favorite pet is Rabbit.")
26          case 4:
27              fmt.Println("Your favorite pet is Turtle.")
28          case 5:
29              fmt.Println("Your favorite pet is Fish.")
30          case 6:
31              fmt.Println("Your favorite pet is not in the choices.")
32          default:
33              fmt.Println("You don't have a favorite pet.")
34      }
35  }
```

```
What's your favorite pet?
[1] Dog
[2] Cat
[3] Rabbit
[4] Turtle
[5] Fish
[6] Not in the choices

Enter your choice: 2
Your favorite pet is Cat.
What's your favorite pet?
[1] Dog
[2] Cat
[3] Rabbit
[4] Turtle
[5] Fish
[6] Not in the choices

Enter your choice: 99
You don't have a favorite pet.
```

برنامه بالا به شما اجازه انتخاب حیوان مورد علاقه‌تان را می‌دهد. به اسم هر حیوان یک عدد نسبت داده شده است. شما عدد را وارد می‌کنید و این عدد در دستور switch با مقادیر case مقایسه می‌شود و با هر کدام از آن مقادیر که برابر بود پیغام مناسب نمایش داده خواهد شد. اگر هم با هیچ کدام از مقادیر case ها برابر نبود دستور default اجرا می‌شود. همانطور که قبلاً ذکر شد دستور switch معادل دستور if…else if است. برنامه بالا را به صورت زیر نیز می‌توان نوشت :

```
if (choice == 1) {
```

```
    fmt.Println("Your favorite pet is Dog.")
} else if (choice == 2) {
    fmt.Println("Your favorite pet is Cat.")
} else if (choice == 3) {
    fmt.Println("Your favorite pet is Rabbit.")
} else if (choice == 4) {
    fmt.Println("Your favorite pet is Turtle.")
} else if (choice == 5) {
    fmt.Println("Your favorite pet is Fish.")
} else if (choice == 6) {
    fmt.Println("Your favorite pet is not in the choices.")
} else {
    fmt.Println("You don't have a favorite pet.")
}
```

کد بالا دقیقاً نتیجه‌ای مانند دستور switch دارد. دستور default معادل دستور else می‌باشد. حال از بین این دو دستور (switch و if else) کدامیک را انتخاب کنیم. از دستور switch موقعی استفاده می‌کنیم که مقداری که می‌خواهیم با دیگر مقادیر مقایسه شود ثابت باشد.

تکرار

ساختارهای تکرار، به شما اجازه می‌دهند که یک یا چند دستور کد را، تا زمانی که یک شرط برقرار است، تکرار کنید. بدون ساختارهای تکرار، شما مجبورید همان تعداد کدها را بنویسید، که بسیار خسته کننده است. مثلاً شما مجبورید ۱۰ بار جمله "Hello World." را تایپ کنید مانند مثال زیر :

```
fmt.Println("Hello World!")
fmt.Println("Hello World!")
fmt.Println("Hello World!")
fmt.Println("Hello World!")
fmt.Println("Hello World!")
fmt.Println("Hello World!")
fmt.Println("Hello World!")
fmt.Println("Hello World!")
fmt.Println("Hello World!")
fmt.Println("Hello World!")
```

البته شما می‌توانید با کپی کردن این تعداد کد را راحت بنویسید ولی این کار در کل کیفیت کدنویسی را پایین می‌آورد. راه بهتر برای نوشتن کدهای بالا استفاده از حلقه است. راه بهتر برای نوشتن کدهای بالا استفاده از حلقه است. Go تنها یک دستور تکرار دارد و آن حلقه for است. در درس بعد در مورد استفاده از حلقه for بیشتر توضیح می دهیم.

حلقه for

حلقه for تنها ساختار تکرار در زبان برنامه نویسی Go می باشد. ساختار حلقه for به صورت زیر است :

```
for initialization; condition; operation {
   code to repeat
}
```

مقدار دهی اولیه (initialization) اولین مقداری است که به شمارنده حلقه می‌دهیم. شمارنده فقط در داخل حلقه for قابل دسترسی است. شرط (condition) در اینجا مقدار شمارنده را با یک مقدار دیگر مقایسه می‌کند و تعیین می‌کند که حلقه ادامه یابد یا نه. عملگر (operation) که مقدار اولیه متغیر را کاهش یا افزایش می‌دهد. در زیر یک مثال از حلقه for آمده است:

```
1  package main
2
3  import "fmt"
4
5  func main() {
6      var i, j int
7
8      for i = 1; i <= 4; i++ {
9          for j = 1; j <= 5; j++ {
10             fmt.Print("*")
11         }
12         fmt.Println()
13     }
14 }
```

```
Number 1
Number 2
Number 3
Number 4
Number 5
Number 6
Number 7
Number 8
Number 9
Number 10
```

برنامه بالا اعداد ۱ تا ۱۰ را با استفاده از حلقه for می‌شمارد. ابتدا یک متغیر به عنوان شمارنده تعریف می‌کنیم و آن را با مقدار ۱ مقدار دهی اولیه می‌کنیم. سپس با استفاده از شرط آن را با مقدار ۱۰ مقایسه می‌کنیم که آیا کمتر است یا مساوی؟ توجه کنید که قسمت سوم حلقه (i++) فوراً اجرا نمی‌شود. کد اجرا می‌شود و ابتدا رشته Number و سپس مقدار جاری i یعنی ۱ را چاپ می‌کند. آنگاه یک واحد به مقدار i اضافه شده و مقدار i برابر ۲ می‌شود و بار دیگر i با عدد ۱۰ مقایسه می‌شود و این حلقه تا زمانی که مقدار شرط true شود ادامه می‌یابد. حال اگر بخواهید معکوس برنامه بالا را پیاده سازی کنید یعنی اعداد از بزرگ به کوچک چاپ شوند باید به صورت زیر عمل کنید :

```
for  i = 10; i > 0; i-- {
   //code omitted
}
```

کد بالا اعداد را از ۱۰ به ۱ چاپ می‌کند (از بزرگ به کوچک). مقدار اولیه شمارنده را ۱۰ می‌دهیم و با استفاده از عملگر کاهش (–) برنامه‌ای که شمارش معکوس را انجام می‌دهد ایجاد می‌کنیم. Go به شما اجازه می‌دهد که از حلقه‌ها به صورت تو در تو استفاده کنید. اگر یک حلقه در داخل حلقه دیگر قرار بگیرد، به آن حلقه تو در تو گفته می‌شود. در این نوع حلقه‌ها، به ازای اجرای یک بار حلقه بیرونی، حلقه داخلی به طور کامل اجرا می‌شود. در زیر نحوه ایجاد حلقه تو در تو آمده است :

```
1  package main
2
3  import "fmt"
4
5  func main() {
6
7      var i, j int
8
9      for i = 1; i <= 4; i++ {
10         for j = 1; j <= 5; j++ {
11             fmt.Print("*")
12         }
13         fmt.Println()
14     }
15
16 }
```

```
*  *  *  *  *
*  *  *  *  *
*  *  *  *  *
*  *  *  *  *
```

در کد بالا به ازای یک بار اجرای حلقه for اول (خط ۹)، حلقه for دوم (10-13) به طور کامل اجرا می‌شود. یعنی وقتی مقدار i برابر عدد ۱ می‌شود، علامت * توسط حلقه دوم ۵ بار چاپ می‌شود، وقتی i برابر ۲ می‌شود، دوباره علامت * پنج بار چاپ می‌شود و ... . در کل منظور از دو حلقه for این است که در ۴ سطر علامت * در ۵ ستون چاپ شود یا ۴ سطر ایجاد شود و در هر سطر ۵ بار علامت * چاپ شود. خط ۱۳ هم برای ایجاد خط جدید است. یعنی وقتی حلقه داخلی به طور کامل اجرا شد، یک خط جدید ایجاد می‌شود و علامت‌های * در خطوط جدید چاپ می‌شوند.

خارج شدن از حلقه با استفاده از break و continue

گاهی اوقات با وجود درست بودن شرط می‌خواهیم حلقه متوقف شود. سؤال اینجاست که چطور این کار را انجام دهید؟ با استفاده از کلمه کلیدی break حلقه را متوقف کرده و با استفاده از کلمه کلیدی continue می‌توان بخشی از حلقه را رد کرد و به مرحله بعد رفت.

برنامه زیر نحوه استفاده از continue و break را نشان می‌دهد :

```go
package main

import "fmt"

func main() {

    var x int
    fmt.Println("Demonstrating the use of break\n")

    for  x = 1; x < 10; x++ {
        if x == 5 {
            break
        }
        fmt.Printf("Number %d \n", x)
    }

    fmt.Println("\nDemonstrating the use of continue\n")

    for x = 1; x < 10; x++ {
        if x == 5 {
            continue
        }
        fmt.Printf("Number %d \n", x)
    }

}
```

```
Demonstrating the use of break.

Number 1
Number 2
Number 3
Number 4

Demonstrating the use of continue.

Number 1
Number 2
Number 3
Number 4
Number 6
Number 7
Number 8
```

```
Number 9
```

در این برنامه از حلقه for برای نشان دادن کاربرد دو کلمه کلیدی فوق استفاده شده است. همانطور که در شرط برنامه (خط ۱۱) آمده است، وقتی که مقدار x به عدد ۵ برسد، سپس دستور break اجرا (خط ۱۲) و حلقه بلافاصله متوقف می‌شود، حتی اگر شرط x < 10 برقرار باشد. از طرف دیگر در خط ۲۰ حلقه for فقط برای یک تکرار خاص متوقف شده و سپس ادامه می‌یابد. (وقتی مقدار x برابر ۵ شود حلقه از ۵ رد شده و مقدار ۵ را چاپ نمی‌کند و بقیه مقادیر چاپ می‌شوند).

آرایه

آرایه نوعی متغیر است که لیستی از آدرسهای مجموعه‌ای از داده‌های هم نوع را در خود ذخیره می‌کند. تعریف چندین متغیر از یک نوع برای هدفی یکسان بسیار خسته کننده است. مثلاً اگر بخواهید صد متغیر از نوع اعداد صحیح تعریف کرده و از آنها استفاده کنید. مطمئناً تعریف این همه متغیر بسیار کسالت آور و خسته کننده است. اما با استفاده از آرایه می‌توان همه آنها را در یک خط تعریف کرد. در زیر راهی ساده برای تعریف یک آرایه نشان داده شده است :

```
var arrayName [size] datatype
```

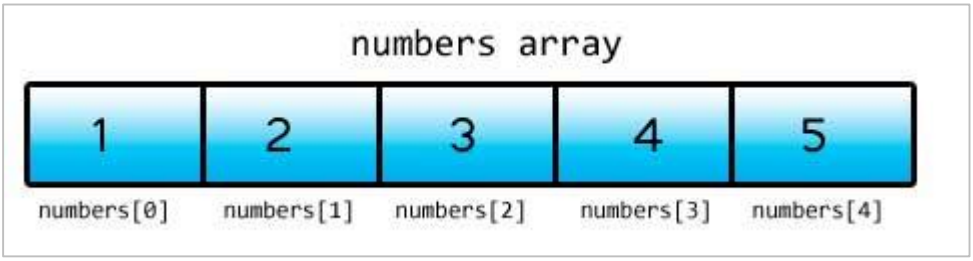
arrayName که نام آرایه را نشان می‌دهد. هنگام نامگذاری آرایه بهتر است که نام آرایه نشان دهنده نوع آرایه باشد. به عنوان مثال برای نامگذاری آرایه‌ای که اعداد را در خود ذخیره می‌کند از کلمه number استفاده کنید. Size، اندازه آرایه را مشخص می کند و به کامپایلر می‌گوید شما قصد دارید چه تعداد داده یا مقدار را در آرایه ذخیره کنید. Datatype نوع داده‌هایی را نشان می‌دهد که آرایه در خود ذخیره می‌کند. برای تعریف یک آرایه که ۵ مقدار از نوع اعداد صحیح در خود ذخیره می‌کند باید به صورت زیر عمل کنیم :

```
var numbers [5] int
```

در این مثال ۵ آدرس از فضای حافظه کامپیوتر شما برای ذخیره ۵ مقدار رزرو می‌شود. حال چطور مقادیرمان را در هر یک از این آدرسها ذخیره کنیم؟ برای دسترسی و اصلاح مقادیر آرایه از اندیس یا مکان آنها استفاده می‌شود.

```
numbers[0] = 1
numbers[1] = 2
numbers[2] = 3
numbers[3] = 4
numbers[4] = 5
```

اندیس یک آرایه از صفر شروع شده و به یک واحد کمتر از طول آرایه ختم می‌شود. به عنوان مثال شما یک آرایه ۵ عضوی دارید، اندیس آرایه از ۰ تا ۴ می‌باشد چون طول آرایه ۵ است پس ۱-۵ برابر است با ۴. این بدان معناست که اندیس ۰ نشان دهنده اولین عضو آرایه است و اندیس ۱ نشان دهنده دومین عضو و الی آخر. برای درک بهتر مثال بالا به شکل زیر توجه کنید :



به هر یک از اجزاء آرایه و اندیسهای داخل کروشه توجه کنید. کسانی که تازه شروع به برنامه نویسی کرده‌اند معمولاً در گذاشتن اندیس دچار اشتباه می‌شوند و مثلاً ممکن است در مثال بالا اندیسها را از ۱ شروع کنند. یکی دیگر از راه‌های تعریف سریع و مقدار دهی یک آرایه به صورت زیر است :

```
var arrayName = [size] datatype { val1, val2, ... valN }
```

در این روش شما می‌توانید فوراً بعد از تعریف اندازه آرایه مقادیر را در داخل آکولاد قرار دهید. به یاد داشته باشید که هر کدام از مقادیر را با استفاده از کاما از هم جدا کنید. همچنین تعداد مقادیر داخل آکولاد باید با اندازه آرایه تعریف شده برابر باشد. به مثال زیر توجه کنید :

```
var numbers = [5] int { 1, 2, 3, 4, 5 }
```

این مثال با مثال قبل هیچ تفاوتی ندارد و تعداد خطهای کدنویسی را کاهش می‌دهد. شما می‌توانید با استفاده از اندیس به مقدار هر یک از اجزاء آرایه دسترسی یابید و آنها را به دلخواه تغییر دهید. تعداد اجزاء آرایه در مثال بالا ۵ است و ما ۵ مقدار را در آن قرار می‌دهیم. اگر تعداد مقادیری که در آرایه قرار می‌دهیم کمتر یا بیشتر از طول آرایه باشد با خطا مواجه می‌شویم. نکته ای که بهتر است در اینجا به آن اشاره شود این است که طول آرایه ثابت و غیر قابل تغییر می باشد. به کد زیر توجه کنید:

```
1  package main
2
3  import "fmt"
4
5  func main() {
6
7      var numbers = [5] int { 1, 2, 3, 4, 5 }
```

```
8
9       numbers[5] = 6
10
11      fmt.Println(numbers)
12
13  }
```

در کد بالا اگر بخواهیم عدد ۶ را به آخر آرایه اضافه کنیم، با خطای out of bounds مواجه می شویم. چون ما در خط ۷ گفته ایم که تعداد عناصر آرایه ۵ است و در نتیجه کامپایلر اجازه اضافه کردن عنصر جدید را نمی دهد.

دستیابی به مقادیر آرایه با استفاده از حلقه for

در زیر مثالی در مورد استفاده از آرایه‌ها آمده است. در این برنامه ۵ مقدار از کاربر گرفته شده و میانگین آنها حساب می‌شود:

```
1   package main
2
3   import "fmt"
4
5   func main() {
6
7       var numbers [5] int
8       var total       int = 0
9       var average     float32
10      var i           int
11
12      for  i = 0; i < len(numbers); i++ {
13          fmt.Print("Enter a number: ")
14          fmt.Scanln(&numbers[i])
15      }
16
17      for i = 0; i < len(numbers); i++ {
18          total += numbers[i];
19      }
20
21      average = (float32(total / len(numbers)))
22
23      fmt.Printf("Average = %.2f", average)
24
25  }
```

```
Enter a number: 90
Enter a number: 85
Enter a number: 80
Enter a number: 87
Enter a number: 92
Average = 86
```

در خط ۷ یک آرایه تعریف شده است که می‌تواند ۵ عدد صحیح را در خود ذخیره کند. خطوط ۸ و ۹ متغیرهایی تعریف شده‌اند که از آنها برای محاسبه میانگین استفاده می‌شود. توجه کنید که مقدار اولیه total صفر است تا از بروز خطا هنگام اضافه شدن مقدار به آن جلوگیری شود. در خطوط ۱۲ تا ۱۵ حلقه for برای تکرار و گرفتن ورودی از کاربر تعریف شده است. از متد طول (len()) آرایه برای تشخیص تعداد اجزای آرایه استفاده می‌شود. اگر چه می‌توانستیم به سادگی در حلقه for مقدار ۵ را برای شرط قرار دهیم ولی استفاده از خاصیت طول آرایه کار راحت‌تری است و می‌توانیم طول آرایه را تغییر دهیم و شرط حلقه for با تغییر جدید هماهنگ می‌شود. در خط ۱۴ ورودی دریافت شده از کاربر در آرایه ذخیره می‌شود. اندیس استفاده شده در number (خط ۱۴) مقدار i جاری در حلقه است. برای مثال در ابتدای حلقه مقدار i صفر است بنابراین وقتی در خط ۱۴ اولین داده از کاربر گرفته می‌شود اندیس آن برابر صفر می‌شود. در تکرار بعدی i یک واحد اضافه می‌شود و در نتیجه در خط ۱۴ و بعد از ورود دومین داده توسط کاربر اندیس آن برابر یک می‌شود. این حالت تا زمانی که شرط در حلقه for برقرار است ادامه می‌یابد. در خطوط ۱۹-۱۷ از حلقه for دیگر برای دسترسی به مقدار هر یک از داده‌های آرایه استفاده شده است. در این حلقه نیز مانند حلقه قبل از مقدار متغیر شمارنده به عنوان اندیس استفاده می‌کنیم.

هر یک از اجزای عددی آرایه به متغیر total اضافه می‌شوند. بعد از پایان حلقه می‌توانیم میانگین اعداد را حساب کنیم (خط ۲۱). مقدار total را بر تعداد اجزای آرایه (تعداد عددها) تقسیم می‌کنیم. برای دسترسی به تعداد اجزای آرایه می‌توان از متد len() آرایه استفاده کرد. توجه کنید که در اینجا ما حاصل از تقسیم را به float32 تبدیل کرده‌ایم بنابراین نتیجه عبارت یک مقدار از نوع float32 خواهد شد و دارای بخش کسری می‌باشد. خط ۲۳ مقدار میانگین را در صفحه نمایش چاپ می‌کند. طول آرایه بعد از مقدار دهی نمی‌تواند تغیر کند. به عنوان مثال اگر یک آرایه را که شامل ۵ جز است مقدار دهی کنید دیگر نمی‌توانید آن را مثلاً به ۱۰ جز تغییر اندازه دهید. البته تعداد خاصی از کلاسها مانند آرایه‌ها عمل می‌کنند و توانایی تغییر تعداد اجزای تشکیل دهنده خود را دارند. آرایه‌ها در برخی شرایط بسیار پر کاربرد هستند و تسلط شما بر این مفهوم و اینکه چطور از آنها استفاده کنید بسیار مهم است.

آرایه های چند بعدی

آرایه‌های چند بعدی آرایه‌هایی هستند که برای دسترسی به هر یک از عناصر آنها باید از چندین اندیس استفاده کنیم. یک آرایه چند بعدی را می‌توان مانند یک جدول با تعدای ستون و ردیف تصور کنید. با افزایش اندیسها اندازه ابعاد آرایه نیز افزایش می‌یابد و آرایه‌های چند بعدی با بیش از دو اندیس به وجود می‌آیند. نحوه ایجاد یک آرایه با دو بعد به صورت زیر است :

```
var arrayName [row] [column] dataType
```

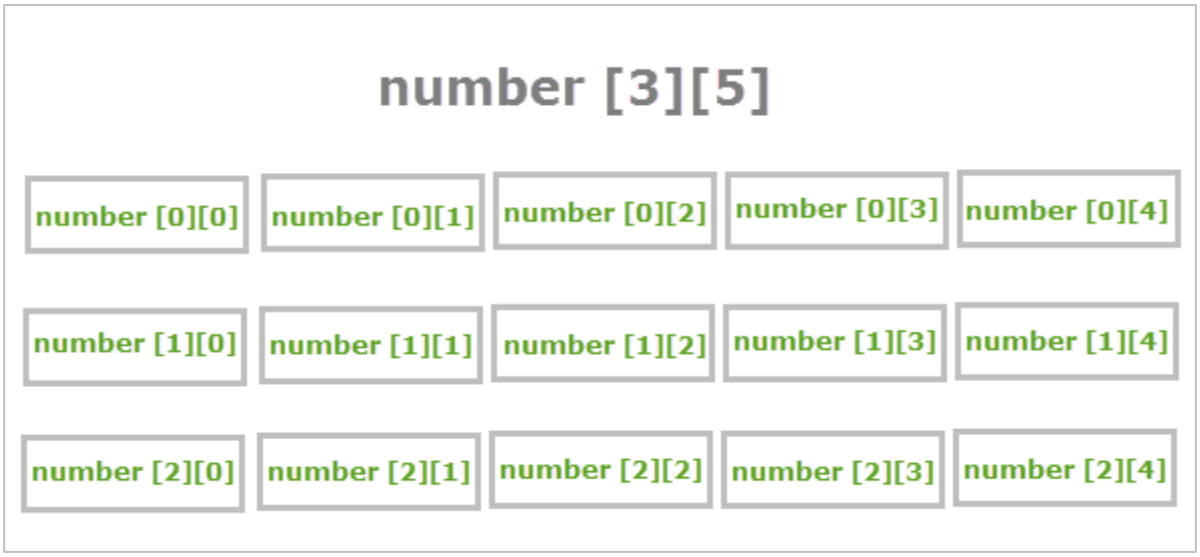
و یک آرایه سه بعدی به صورت زیر ایجاد می‌شود :

```
var arrayName [row] [column] [height] dataType
```

به دلیل اینکه آرایه‌های سه بعدی یا آرایه‌های با بیشتر از دو بعد بسیار کمتر مورد استفاده قرار می‌گیرند اجازه بدهید که در این درس بر روی آرایه‌های دو بعدی تمرکز کنیم. اگر آرایه دو بعدی را یک مربع یا مستطیل فرض کنیم، در تعریف این نوع آرایه ابتدا نام آرایه، سپس تعداد سطر (row) و ستون های (column) آن و در آخر نوع داده آرایه (dataType) را مشخص می کنیم. در یک آرایه دو بعدی برای دسترسی به هر یک از عناصر به دو مقدار نیاز داریم یکی مقدار X و دیگری مقدار Y که مقدار x نشان دهنده ردیف و مقدار Y نشان دهنده ستون آرایه است البته اگر ما آرایه دو بعدی را به صورت جدول در نظر بگیریم. یک آرایه سه بعدی را می‌توان به صورت یک مکعب تصور کرد که دارای سه بعد است و x طول، Y عرض و z ارتفاع آن است. یک مثال از آرایه دو بعدی در زیر آمده است :

```
var numbers [3][5] int
```

کد بالا به کامپایلر می‌گوید که فضای کافی به عناصر آرایه اختصاص بده)در این مثال ۱۵ خانه). در شکل زیر مکان هر عنصر در یک آرایه دو بعدی نشان داده شده است.



مقدار ۳ را به x اختصاص می‌دهیم چون ۳ سطر و مقدار ۵ را به ۷ چون ۵ ستون داریم اختصاص می‌دهیم. چطور یک آرایه چند بعدی را مقدار دهی کنیم؟ چند راه برای مقدار دهی به آرایه‌ها وجود دارد. یک راه این است که مقادیر عناصر آرایه را در همان زمان تعریف آرایه، مشخص کنیم :

```
var numbers = [3][5] int {
    { 1, 2, 3, 4, 5     },
    { 6, 7, 8, 9, 10    },
    { 11, 12, 13, 14, 15},
}
```

و یا می‌توان مقدار دهی به عناصر را به صورت دستی انجام داد مانند :

```
array[0][0] = value
array[0][1] = value
array[0][2] = value
array[1][0] = value
array[1][1] = value
array[1][2] = value
array[2][0] = value
array[2][1] = value
array[2][2] = value
```

همانطور که مشاهده می‌کنید برای دسترسی به هر یک از عناصر در یک آرایه دو بعدی به سادگی می‌توان از اندیسهای X و Y و یک جفت کروشه مانند مثال استفاده کرد.

گردش در میان عناصر آرایه‌های چند بعدی

گردش در میان عناصر آرایه‌های چند بعدی نیاز به کمی دقت دارد. یکی از راههای آسان استفاده حلقه for تو در تو است. برنامه زیر نشان می‌دهد که چطور از حلقه for برای خواندن همه مقادیر آرایه و تعیین انتهای ردیف‌ها استفاده کنید:

```
1  package main
2
3  import "fmt"
4
5  func main() {
6
7      var row, column int
8      var numbers = [3][5] int{
9          { 1, 2, 3, 4, 5      },
10         { 6, 7, 8, 9, 10     },
11         { 11, 12, 13, 14, 15 },
12     }
13
14     for row=0; row<len(numbers); row++ {
15         for column=0; column<len(numbers[0]); column++ {
```

```
16              fmt.Printf("%d  ", numbers[row][column])
17          }
18          fmt.Println()
19      }
20  }
```

```
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
```

همانطور که در مثال بالا نشان داده شده است با استفاده از یک حلقه ساده for نمی‌توان به تمامی مقادیر دسترسی یافت بلکه، به یک حلقه for تو در تو نیاز داریم. در اولین حلقه for (خط ۱۴) یک متغیر تعریف شده است که در میان ردیف‌های آرایه (row) گردش می‌کند. این حلقه تا زمانی ادامه می‌یابد که مقدار ردیف کمتر از طول اولین بعد باشد. در این مثال از متد ()len استفاده کرده‌ایم. این متد طول آرایه را در یک بعد خاص نشان می‌دهد. به عنوان مثال برای به دست آوردن طول اولین بعد آرایه که همان تعداد ردیف‌ها می‌باشد از دستور len(numbers) استفاده کرده‌ایم.

در داخل اولین حلقه for حلقه for دیگری تعریف شده است (خط ۱۵). در این حلقه یک شمارنده برای شمارش تعداد ستونهای (columns) هر ردیف تعریف شده است و در شرط داخل آن بار دیگر از متد ()len استفاده شده است، ولی این بار به نوعی دیگر و به صورت زیر از آن استفاده کرده ایم:

```
len(numbers[column])
```

استفاده از column به عنوان اندیس، به معنای تعداد ستون های آن ردیف است. در نتیجه در حلقه for دوم وقتی برای اولین بار حلقه دوم اجرا می شود، اندیس column برابر ۰ و حاصل عبارت len(numbers[0]) عدد ۵ می شود. با این تفاسیر، وقتی کل برنامه برای اولین بار اجرا می شود، مقدار ردیف 0 (row) شده و حلقه for دوم از [۰][۰] تا [۴][۰] اجرا شود. سپس مقدار هر عنصر از آرایه را با استفاده از حلقه نشان می‌دهیم، اگر مقدار ردیف (row) برابر ۰ و مقدار ستون (col) برابر ۰ باشد مقدار عنصری که در ستون ۱ و ردیف ۱ (numbers[0][0]) قرار دارد نشان داده خواهد شد که در مثال بالا عدد ۱ است. در مجموعه از حلقه for اول برای پیمایش سطرها و از دومی برای پیمایش ستون ها استفاده می شود. در این دو حلقه به جای len(numbers) و len(numbers[column]) می توان به ترتیب اعداد ۳ و ۵ را هم قرار داد.

بعد از اینکه دومین حلقه تکرار به پایان رسید، فوراً دستورات بعد از آن اجرا خواهند شد، که در اینجا دستور fmt.Println() که به برنامه اطلاع می‌دهد که به خط بعد برود. سپس حلقه با اضافه کردن یک واحد به مقدار row این فرایند را دوباره تکرار می‌کند. سپس دومین

حلقه for اجرا شده و مقادیر دومین ردیف نمایش داده می‌شود. این فرایند تا زمانی اجرا می‌شود که مقدار row کمتر از طول اولین بعد باشد. حال بیایید آنچه را از قبل یاد گرفته‌ایم در یک برنامه به کار بریم. این برنامه نمره چهار درس مربوط به سه دانش آموز را از ما می‌گیرد و معدل سه دانش آموز را حساب می‌کند.

```go
package main

import "fmt"

func main() {

    var student, grade       int
    var studentGrades [3][4] float32
    var total                float32

    for student = 0; student < len(studentGrades); student++ {
        total = 0
        fmt.Printf("\nEnter grades for Student %d \n", student+1)

        for grade = 0; grade < len(studentGrades[grade]); grade++ {
            fmt.Printf("Enter Grade #%d: ", grade+1)
            fmt.Scanln(&studentGrades[student][grade])
            total += studentGrades[student][grade]
        }

        fmt.Printf("Average is %.2f", (total / float32(len(studentGrades[student]))))
        fmt.Println();
    }

}
```

```
Enter grades for Student 1
Enter Grade #1: 92
Enter Grade #2: 87
Enter Grade #3: 89
Enter Grade #4: 95
Average is 90.75

Enter grades for Student 2
Enter Grade #1: 85
Enter Grade #2: 85
Enter Grade #3: 86
Enter Grade #4: 87
Average is 85.75

Enter grades for Student 3
Enter Grade #1: 90
Enter Grade #2: 90
Enter Grade #3: 90
Enter Grade #4: 90
Average is 90.00
```

در برنامه بالا یک آرایه چند بعدی از نوع float32 تعریف شده است (خط ۸). همچنین یک متغیر به نام total تعریف می‌کنیم که مقدار محاسبه شده معدل هر دانش آموز را در آن قرار دهیم. حال وارد حلقه for تو در تو می‌شویم (خط ۱۱). در اولین حلقه for یک متغیر به نام sudent برای تشخیص پایه درسی هر دانش آموز تعریف کرده‌ایم. از متد ()len هم برای تشخیص تعداد دانش آموزان استفاده شده است. وارد بدنه حلقه for می‌شویم. در خط ۱۲ مقدار متغیر total را برابر صفر قرار می‌دهیم. بعداً مشاهده می‌کنید که چرا این کار را انجام دادیم. سپس برنامه یک پیغام را نشان می‌دهد و از شما می‌خواهد که شماره دانش آموز را وارد کنید (student + 1). عدد ۱ را به student اضافه کرده‌ایم تا به جای نمایش Student 0، با Student 1 شروع شود، تا طبیعی‌تر به نظر برسد. سپس به دومین حلقه for در خط ۱۵ می‌رسیم. در این حلقه یک متغیر شمارنده به نام grade تعریف می‌کنیم که طول دومین بعد آرایه را با استفاده از len(studentGrades[grade]) به دست می‌آورد. این طول تعداد نمراتی را که برنامه از سؤال می‌کند را نشان می‌دهد. برنامه چهار نمره مربوط به دانش آموز را می‌گیرد. هر وقت که برنامه یک نمره را از کاربر دریافت می‌کند، نمره به متغیر total اضافه می‌شود.

وقتی همه نمره‌ها وارد شدند، متغیر total هم جمع همه نمرات را نشان می‌دهد. در خط ۲۱ معدل دانش آموز نشان داده می‌شود. معدل از تقسیم کردن total (جمع) بر تعداد نمرات به دست می‌آید. از len(studentGrades[student]) هم برای به دست آوردن تعداد نمرات استفاده می‌شود.

Slice

Slice در واقع همان آرایه است با این تفاوت که در زمان تعریف برای آن طول تعیین نمی کنیم. این کار باعث قدرت و انعطاف بیشتر می شود. در زیر روش های ایجاد Slice ذکر شده اند:

```
make([]Type, length, capacity)

make([]Type, length)

[]Type{}

[]Type{value1, value2, ..., valueN}
```

همانطور که در درس قبل ذکر شد آرایه ها دارای طول ثابتی هستند و بعد از تعریف نمی توان مقداری به آنها اضافه کرد. ولی Slice ها این مشکل را برطرف کرده اند. به مثال زیر توجه کنید:

```
1  package main
2
3  import "fmt"
4
5  func main() {
6      var numbers = [] int { 10, 15, 17, 20, 13}
```

```
7
8       numbers = append(numbers, 6)
9
10      fmt.Println(numbers)
11  }
```

```
[10  15  17  20  13]
```

در خط ۶ کد بالا، یک Slice تعریف کرده ایم و در خط ۸ به راحتی مقدار ۶ را به آن اضافه کرده ایم. Slice دارای متدهای مختلفی جهت حذف، اضافه، تکه تکه کردن و ... عناصر را می دهند. یکی از آنها متد append() است که در کد از آن برای اضافه کردن یک عنصر به Slice استفاده کرده ایم. روش دیگر برای ایجاد Slice استفاده از متد make() است. این متد سه پارامتر می گیرد. اولین پارامتر نوع داده هایی که Slice قرار است بگیرد، دومی طول و سومی ظرفیت Slice می باشد. به کد زیر توجه کنید:

```
1   package main
2
3   import "fmt"
4
5   func main() {
6       numbers := make([]int, 5)
7
8       numbers[0] = 10
9       numbers[1] = 15
10      numbers[2] = 17
11      numbers[3] = 20
12      numbers[4] = 13
13
14      fmt.Println(numbers)
15  }
```

```
[13 15 17 20 13]
```

در کد بالا یک Slice به نام numbers و با طول ۵ تعریف کرده ایم. حال اگر بخواهیم، می توانیم با استفاده از کد زیر یک عنصر دیگر به ین Slice اضافه کنیم:

```
numbers = append(numbers, 50)
```

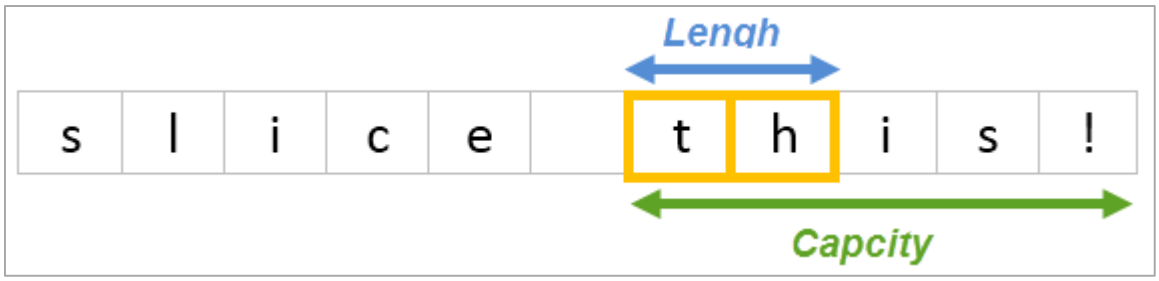
اما یک نکته، در خط ۶ کد بالا، ما طول Slice را مشخص کرده ایم. حال تکلیف پارامتر سوم یا ظرفیت Slice چه می شود و اصلا تفاوت این دو با هم چیست؟ به مثال زیر توجه کنید:

```
1   package main
2
```

```go
import "fmt"

func main() {
    var arrayOfCharacter = [11] string { "s","l","i","c","e"," ","t","h","i","s","!" }

    sliceOfCharacter := arrayOfCharacter[6:8]

    fmt.Println("Capacity of array :", cap(arrayOfCharacter))
    fmt.Println("Lenght   of array :", cap(arrayOfCharacter))
    fmt.Println()
    fmt.Println("Capacity of slice :", cap(sliceOfCharacter))
    fmt.Println("Lenght   of slice :", len(sliceOfCharacter))
}
```

```
Capacity of array : 11
Lenght   of array : 11

Capacity of slice : 7
Lenght   of slice : 3
```

در خط ۶ کد بالا، ما یک آرایه به طول ۱۱ کاراکتر ایجاد کرده ایم. یعنی تعداد عناصر آرایه ۱۱ می باشد. در خط ۸ یک برش یا Slice از این آرایه ایجاد کرده ایم. [۴:۷] در خط ۱۱ بدین معنی است که از عنصری که دارای اندیس ۶ تا عنصری که دارای اندیس ۷ است را برش بزن. یعنی کاراکتر های t و h را جدا کن و در داخل یک Slice به نام sliceOfCharacter قرار بده. حال در خطوط ۱۴-۱۰ با استفاده از دو متد len() و cap() طول و ظرفیت آرایه و Slice را به دست آورده ایم. طول و ظرفیت آرایه به هم برابرند. یعنی تعداد عناصر آرایه مشخص کننده طول و ظرفیت آرایه است. اما در Slice وضع بدین منوال نیست. طول Slice تعداد عناصر Slice، که در مثال بالا ۲ عنصر هستند، ولی ظرفیت Slice از جاییکه برش آرایه شروع شده است تا پایان آرایه می باشد یعنی ۵. برای درک بهتر به شکل زیر توجه کنید:



map

از map زمانی استفاده می‌شود که بخواهید اطلاعات را بر اساس کلید/ مقدار ذخیره کنید. به عنوان مثال نام دانش آموز و نمره او در امتحان. برای ایجاد یک map از تابع داخلی ()make به صورت زیر استفاده می‌شود:

```
map_variable = make(map[key_data_type]value_data_type)
```

به مثال زیر توجه کنید:

```
students := make(map[string]int)

students["Jenny"]     = 87
students["Peter"]     = 70
students["Mary Jane"] = 64
students["Azhar"]     = 79
```

در مثال بالا یک map ایجاد کرده‌ایم که نوع کلیدهای آن رشته‌ای (string) و مقادیر آن عددی (int) است. map بالا را به صورت زیر هم می‌توان ایجاد کرد:

```
students := map[string]int {
    "Jenny"    : 87,
    "Peter"    : 10,
    "Mary Jane": 64,
    "Azhar"    : 79,
}
```

در روش بالا متد ()make حذف می‌شود و بین کلید/مقدارها علامت: و بین هر دو کلید مقدار علامت, قرار می‌گیرد. برای چاپ مقدار یک کلید مثلاً Azhar هم به صورت زیر عمل می‌شود:

```
fmt.Println(students["Azhar"])
```

```
79
```

یعنی ابتدا نامی که برای map انتخاب کرده‌اید را نوشته و سپس در داخل براکت نام کلید را می‌نویسید. برای چاپ تمام کلید/مقدارها کافیست فقط نام map را بنویسید:

```
fmt.Println(students)
```

```
map[Peter:10 Mary Jane:64 Azhar:79 Jenny:87]
```

برای حذف یک مقدار هم می‌توان از تابع ()delete به صورت زیر استفاده کرد:

```
delete(students, "Peter")
```

این متد دو آرگومان می‌گیرد، اولی نام map و دیگری کلیدی که قرار است حذف شود.

Range

از کلمه کلیدی range در داخل حلقه for و برای پیمایش و به دست آوردن مقادیر موجود در آرایه، Slice و map استفاده می‌شود. به مثال زیر توجه کنید :

```
package main

import "fmt"

func main() {
    fmt.Println("Array example ...")

    numbers := []int {7, 13, 2, 5, 9, 5, 4, 8, 10}
    for i:= range numbers {
        fmt.Print(numbers[i], " ")
    }


    fmt.Println("\n")

    students := map[string]int{
        "Jenny"    : 87,
        "Peter"    : 10,
        "Mary Jane": 64,
        "Azhar"    : 79,
    }

    fmt.Println("Map example with key/value")

    for key, value := range students {
        fmt.Printf("%s: %d \n", key, value)
    }

    fmt.Println("\nMap example just value")

    for _, value := range students {
        fmt.Printf("%d \n", value)
    }
}
```

```
Array Example ...
7 13 2 5 9 5 4 8 10

map example with key/value
```

```
Peter: 10
Mary Jane: 64
Azhar: 79
Jenny: 87

map example just value
87
10
64
79
```

در خط ۸ یک Slice یا آرایه تعریف شده است. همانطور که قبلاً گفته شد برای دسترسی به مقادیر موجود در یک Slice یا آرایه به اندیس آن مقدار نیاز داریم. در خط ۹، اندیس تک تک عناصر موجود در Slice را در هر بار تکرار حلقه for گرفته و در داخل i قرار می‌دهیم. سپس در خط ۱۰ از این اندیس در داخل کروشه و جلوی نام Slice استفاده می‌کنیم. یعنی مثلاً در اولین اجرای حلقه for، اندیس ۰ و در نتیجه مقدار اولین عنصر آرایه با استفاده از numbers[0] به دست آمده و چاپ می‌شود. در خطوط ۱۶-۲۱ یک map به نام students تعریف شده است. یکی از قابلیت‌های range در داخل حلقه for این است که می‌توان، اندیس و مقدارهای آرایه، Slice و map را هم به دست آورد. در خط ۲۵ این کار را انجام داده‌ایم و دو متغیر تعریف کرده و نتیجه برنامه به طور خودکار اندیس‌های map را در داخل اولین متغیر یعنی Key و مقادیر را در داخل متغیر دوم یعنی Value قرار می‌دهد. حال اگر بخواهیم فقط مقادیر را چاپ کنیم کافیست که از علامت (blank identifier) _ استفاده کنیم. کاری که در خط ۳۱ انجام داده‌ایم. این علامت در اینجا بدین معنی است که ما نیازی به کلیدها نداریم.

متد

متدها به شما اجازه می‌دهند که یک رفتار یا وظیفه را تعریف کنید و مجموعه‌ای از کدها هستند که در هر جای برنامه می‌توان از آنها استفاده کرد. متدها دارای آرگومانهایی هستند که وظیفه متد را مشخص می‌کنند. نمی‌توان یک متد را در داخل متد دیگر تعریف کرد. وقتی که شما در برنامه یک متد را صدا می‌زنید، برنامه به قسمت تعریف متد رفته و کدهای آن را اجرا می‌کند. در Go متدی وجود دارد که نقطه آغاز هر برنامه است و بدون آن برنامه‌ها نمی‌دانند باید از کجا شروع شوند، این متد main() نام دارد.

پارامترها همان چیزهایی هستند که متد منتظر دریافت آنها است.

آرگومان‌ها مقادیری هستند که به پارامترها ارسال می‌شوند.

گاهی اوقات دو کلمه پارامتر و آرگومان به یک منظور به کار می‌روند. ساده‌ترین ساختار یک متد به صورت زیر است :

```
func MethodName(Parameter List) returnType {
   code to execute
}
```

به برنامه ساده زیر توجه کنید. در این برنامه از یک متد برای چاپ یک پیغام در صفحه نمایش استفاده شده است :

```
1  package main
2
3  import "fmt"
4
5  func PrintMessage() {
6      fmt.Println("Hello World!")
7  }
8
9  func main() {
10     PrintMessage()
11 }
```

```
Hello World!
```

در خطوط ۷-۵ یک متد تعریف کرده‌ایم. می‌توانید این متد را، زیر متد main() تعریف کنید. می‌توان این متد را در داخل متد دیگر صدا زد (فراخوانی کرد). متد دیگر ما در اینجا متد main() است که می‌توانیم در داخل آن نام متدی که برای چاپ یک پیغام تعریف کرده‌ایم (یعنی متد PrintMessage()) را صدا بزنیم.

در تعریف متد بالا قسمت returnType را مشخص نکرده ایم، که نشان دهنده آن است که متد مقدار برگشتی ندارد. در درس آینده در مورد مقدار برگشتی از یک متد و استفاده از آن برای اهداف مختلف توضیح داده خواهد شد. نام متد ما PrintMessage() است. به این نکته توجه کنید که در نامگذاری متد از روش پاسکال (حرف اول هر کلمه بزرگ نوشته می‌شود) استفاده کرده‌ایم. این روش نامگذاری قراردادی است و می‌توان از این روش استفاده نکرد، اما پیشنهاد می‌شود که از این روش برای تشخیص متدها استفاده کنید. بهتر است در نامگذاری متدها از کلماتی استفاده شود که کار آن متد را مشخص می‌کند مثلاً نام‌هایی مانند GoToBed یا OpenDoor. همچنین به عنوان مثال اگر مقدار برگشتی متد یک مقدار بولی باشد می‌توانید اسم متد خود را به صورت یک کلمه سوالی انتخاب کنید مانند IsLeapyear یا IsTeenager… ولی از کذاشتن علامت سؤال در آخر اسم متد خودداری کنید. دو پرانتزی که بعد از نام می‌آید نشان دهنده آن است که نام متعلق به یک متد است. در این مثال در داخل پرانتزها هیچ چیزی نوشته نشده چون پارامتری ندارد. در درسهای آینده در مورد متدها بیشتر توضیح می‌دهیم.

بعد از پرانتزها دو آکولاد قرار می‌دهیم که بدنه متد را تشکیل می‌دهد و کدهایی را که می‌خواهیم اجرا شوند را در داخل این آکولادها می‌نویسیم. در داخل متد main() متدی را که در خط ۵ ایجاد کرده‌ایم را صدا می‌زنیم. برای صدا زدن یک متد کافیست نام آن را نوشته و بعد از نام پرانتزها را قرار دهیم.

اگر متد دارای پارامتر باشد، باید شما آرگومانها را به ترتیب در داخل پرانتزها قرار دهید. در این مورد نیز در درسهای آینده توضیح بیشتری می‌دهیم. با صدا زدن یک متد کدهای داخل بدنه آن اجرا می‌شوند. برای اجرای متد PrintMessage() برنامه از متد main() به محل تعریف متد PrintMessage() می‌رود. مثلاً وقتی ما متد PrintMessage() را در خط ۱۰ صدا می‌زنیم برنامه از خط ۱۰ به خط ۵، یعنی جایی که متد تعریف شده می‌رود. اکنون ما یک متد در برنامه داریم و همه متدهای این برنامه می‌توانند آن را صدا بزنند.

مقدار برگشتی از یک متد

متدها می‌توانند مقدار برگشتی از هر نوع داده‌ای داشته باشند. این مقادیر می‌توانند در محاسبات یا به دست آوردن یک داده مورد استفاده قرار بگیرند. در زندگی روزمره فرض کنید که کارمند شما یک متد است و شما او را صدا می‌زنید و از او می‌خواهید که کار یک سند را به پایان برساند. سپس از او می‌خواهید که بعد از اتمام کارش سند را به شما تحویل دهد. سند همان مقدار برگشتی متد است. نکته مهم در مورد یک متد، مقدار برگشتی و نحوه استفاده شما از آن است. برگشت یک مقدار از یک متد آسان است. کافیست در تعریف متد به روش زیر عمل کنید :

```
func MethodName() returnType {
   return value
}
```

returnType در اینجا نوع داده‌ای مقدار برگشتی را مشخص می‌کند (int، bool و ...) در داخل بدنه متد کلمه کلیدی return و بعد از آن یک مقدار یا عبارتی که نتیجه آن یک مقدار است را می‌نویسیم. نوع این مقدار برگشتی باید از انواع ساده بوده و در هنگام نامگذاری متد و بعد از پرانتزها ذکر شود. مثال زیر یک متد که دارای مقدار برگشتی است را نشان می‌دهد.

```
1  package main
2
3  import "fmt"
4
5  func CalculateSum() int {
6      var firstNumber  int = 10
7      var secondNumber int = 5
8
9      var sum = firstNumber + secondNumber
10
11     return sum
```

```
12  }
13
14  func main() {
15      var result = CalculateSum()
16      fmt.Printf("Sum is %d.", result)
17  }
```

```
Sum is 15.
```

همانطور که در خط ۵ مثال فوق مشاهده می‌کنید هنگام تعریف متد از کلمه int استفاده کرده‌ایم که نشان دهنده آن است که متد ما دارای مقدار برگشتی از نوع اعداد صحیح است. در خطوط ۶ و ۷ دو متغیر تعریف و مقدار دهی شده‌اند.

توجه کنید که این متغیرها، متغیرهای محلی هستند. و این بدان معنی است که این متغیرها در سایر متدها مانند متد main() قابل دسترسی نیستند و فقط در متدی که در آن تعریف شده‌اند، قابل استفاده هستند. در خط ۹ جمع دو متغیر در متغیر sum قرار می‌گیرد. در خط ۱۱ مقدار برگشتی sum توسط دستور return فراخوانی می‌شود. در داخل متد main() یک متغیر به نام result در خط ۱۵ تعریف می‌کنیم و متد CalculateSum() را فراخوانی می‌کنیم.

متد CalculateSum() مقدار ۱۵ را بر می‌گرداند که در داخل متغیر result ذخیره می‌شود. در خط ۱۶ مقدار ذخیره شده در متغیر result چاپ می‌شود. متدی که در این مثال ذکر شد متد کاربردی و مفیدی نیست. با وجودیکه کدهای زیادی در متد بالا نوشته شده ولی همیشه مقدار برگشتی ۱۵ است، در حالیکه می‌توانستیم به راحتی یک متغیر تعریف کرده و مقدار ۱۵ را به آن اختصاص دهیم. این متد در صورتی کارامد است که پارامترهایی به آن اضافه شود که در درسهای آینده توضیح خواهیم داد. هنگامی که می‌خواهیم در داخل یک متد از دستور if یا switch استفاده کنیم باید تمام کدها دارای مقدار برگشتی باشند. برای درک بهتر این مطلب به مثال زیر توجه کنید :

```go
package main

import "fmt"

func GetNumber() int {
    var number int
    fmt.Print("Enter a number greater than 10: ")
    fmt.Scanln(&number)

    if number > 10 {
        return number
    } else {
        return 0
    }
}
```

```
func main() {
    var result int = GetNumber()
    fmt.Printf("Result = %d.", result)
}
```

```
Enter a number greater than 10: 11
Result = 11
Enter a number greater than 10: 9
Result = 0
```

در خطوط ۱۵-۵ یک متد با نام GetNumber() تعریف شده است که از کاربر یک عدد بزرگتر از ۱۰ را می‌خواهد. اگر عدد وارد شده توسط کاربر درست نباشد متد مقدار صفر را بر می‌گرداند. و اگر قسمت else دستور if یا دستور return را از آن حذف کنیم در هنگام اجرای برنامه با پیغام خطا مواجه می‌شویم.

چون اگر شرط دستور if نادرست باشد (کاربر مقداری کمتر از ۱۰ را وارد کند) برنامه به قسمت else می‌رود تا مقدار صفر را بر گرداند و چون قسمت else حذف شده است برنامه با خطا مواجه می‌شود و همچنین اگر دستور return حذف شود چون برنامه نیاز به مقدار برگشتی دارد پیغام خطا می‌دهد. و آخرین مطلبی که در این درس می‌خواهیم به شما آموزش دهیم این است که شما می‌توانید از یک متد که مقدار برگشتی ندارد، خارج شوید. حتی اگر متد هیچ نوع برگشتی نداشته باشد، باز هم می‌توانید کلمه کلیدی return را در آن به کار ببرید. استفاده از return باعث خروج از بدنه متد و اجرای کدهای بعد از آن می‌شود.

```
package main

import "fmt"

func TestReturnExit() {
    fmt.Println("Line 1 inside the method TestReturnExit()")
    fmt.Println("Line 2 inside the method TestReturnExit()")

   return

    //The following lines will not execute
    fmt.Println("Line 3 inside the method TestReturnExit()")
    fmt.Println("Line 4 inside the method TestReturnExit()")
}

func main() {
    TestReturnExit();
    fmt.Printf("Hello World!")
}
```

```
Line 1 inside the method TestReturnExit()
Line 2 inside the method TestReturnExit()
```

```
Hello World!
```

در برنامه بالا نحوه خروج از متد با استفاده از کلمه کلیدی return و نادیده گرفتن همه کدهای بعد از این کلمه کلیدی نشان داده شده است. در پایان برنامه متد تعریف شده (TestReturnExit()) در داخل متد main() فراخوانی و اجرا می‌شود. نکته آخر اینکه یک متد می تواند دارای چند مقدار برگشتی باشد. به مثال زیر توجه کنید :

```go
package main

import "fmt"

func ShowMessage(str1, str2 string) (string, string) {
    return str1 , str2
}

func main() {
    fmt.Println(ShowMessage("Hello", " World!"))
}
```

```
Hello  World!
```

همانطور که در کد بالا مشاهده می کنید، برای متدهایی که دارای چند نوع برگشتی هستند باید نوع تمامی مقادیر برگشتی را در داخل پرانتز نوشته و آنها را با کاما از هم جدا کنید، مانند ((string, string)).

پارامترها و آرگومان ها

پارامترها داده‌های خامی هستند که متد آنها را پردازش می‌کند و سپس اطلاعاتی را که به دنبال آن هستید، در اختیار شما قرار می‌دهد. فرض کنید پارامترها مانند اطلاعاتی هستند که شما به یک کارمند می‌دهید که بر طبق آنها کارش را به پایان برساند. یک متد می‌تواند هر تعداد پارامتر داشته باشد. هر پارامتر می‌تواند از انواع مختلف داده باشد. در زیر یک متد با N پارامتر نشان داده شده است :

```go
func MethodName(param1, param2, ... paramN datatype) returnType {
   code to execute
}
```

پارامترها بعد از نام متد و بین پرانتزها قرار می‌گیرند. بر اساس کاری که متد انجام می‌دهد می‌توان تعداد پارامترهای زیادی به متد اضافه کرد. بعد از فراخوانی یک متد باید آرگومانهای آن را نیز تأمین کنید. آرگومان‌ها مقادیری هستند که به پارامترها اختصاص داده می‌شوند. ترتیب ارسال آرگومانها به پارامترها مهم است. عدم رعایت ترتیب در ارسال آرگومانها باعث به وجود آمدن خطای منطقی و خطای زمان اجرا می‌شود. اجازه بدهید که یک مثال بزنیم :

```go
1  package main
```

```
2
3  import "fmt"
4
5  func CalculateSum(number1, number2 int) int {
6      return number1 + number2
7  }
8
9  func main() {
10     var num1, num2 int
11
12     fmt.Print("Enter the first number: ")
13     fmt.Scanln(&num1)
14
15     fmt.Print("Enter the second number: ")
16     fmt.Scanln(&num2)
17
18     fmt.Printf("Sum = %d", CalculateSum(num1, num2))
19 }
```

```
Enter the first number: 10
Enter the second number: 5
Sum = 15
```

در برنامه بالا یک متد به نام `CalculateSum()` (خطوط ۷-۵) تعریف شده است که وظیفه آن جمع مقدار دو عدد است. چون این متد مقدار دو عدد صحیح را با هم جمع می‌کند پس نوع برگشتی ما نیز باید `int` باشد. متد دارای دو پارامتر است که اعداد را به آنها ارسال می‌کنیم. به نوع داده‌ای پارامترها توجه کنید. هر دو پارامتر یعنی `number1` و `number2` مقادیری از نوع اعداد صحیح (`int`) دریافت می‌کنند. در بدنه متد دستور `return` نتیجه جمع دو عدد را بر می‌گرداند. در داخل متد `main()` برنامه از کاربر دو مقدار را درخواست می‌کند و آنها را داخل متغیرها قرار می‌دهد. حال متد را که آرگومانهای آن را آماده کرده‌ایم، فراخوانی می‌کنیم. مقدار `num1` به پارامتر اول و مقدار `num2` به پارامتر دوم ارسال می‌شود. حال اگر مکان دو مقدار را هنگام ارسال به متد تغییر دهیم (یعنی مقدار `num2` به پارامتر اول و مقدار `num1` به پارامتر دوم ارسال شود) هیچ تغییری در نتیجه متد ندارد چون جمع خاصیت جابه جایی دارد.

فقط به یاد داشته باشید که باید ترتیب ارسال آرگومانها هنگام فراخوانی متد دقیقاً با ترتیب قرارگیری پارامترها تعریف شده در متد مطابقت داشته باشد. بعد از ارسال مقادیر ۱۰ و ۵ به پارامترها، پارامترها آنها را دریافت می‌کنند. به این نکته نیز توجه کنید که نام پارامترها طبق قرارداد به شیوه کوهان شتری یا `camelCasing` (حرف اول دومین کلمه بزرگ نوشته می‌شود) نوشته می‌شود. در داخل بدنه متد (خط ۶) دو مقدار با هم جمع می‌شوند و نتیجه به متد فراخوان (متدی که متد `CalculateSum()` را فراخوانی می‌کند) ارسال می‌شود. در درس آینده از یک متغیر برای ذخیره نتیجه محاسبات استفاده می‌کنیم ولی در اینجا مشاهده می‌کنید که می‌توان به سادگی نتیجه جمع را نشان داد (خط ۶). در داخل متد `main()` از ما دو عدد که قرار است با هم جمع شوند درخواست می‌شود.

در خط ۱۹ متد CalculateSum() را فراخوانی می‌کنیم و دو مقدار صحیح به آن ارسال می‌کنیم. دو عدد صحیح در داخل متد با هم جمع شده و نتیجه آنها برگردانده می‌شود. مقدار برگشت داده شده از متد، به وسیله متد Printf() نمایش داده می‌شود (خط ۱۸). در برنامه زیر یک متد تعریف شده است که دارای دو پارامتر از دو نوع داده‌ای مختلف است:

```
1  package main
2
3  import "fmt"
4
5  func ShowMessageAndNumber(message string, number int) {
6      fmt.Println(message)
7      fmt.Printf("Number = %d", number)
8  }
9
10 func main() {
11     ShowMessageAndNumber("Hello World!", 100)
12 }
```

```
Hello World!
Number = 100
```

در مثال بالا یک متدی تعریف شده است که اولین پارامتر آن مقداری از نوع رشته و دومین پارامتر آن مقداری از نوع int دریافت می‌کند. متد به سادگی دو مقداری که به آن ارسال شده است را نشان می‌دهد. در خط ۱۱ متد را اول با یک رشته و سپس یک عدد خاص فراخوانی می‌کنیم. حال اگر متد به صورت زیر فراخوانی می‌شد :

```
ShowMessageAndNumber(100, "Welcome to Gimme Go!")
```

در برنامه خطا به وجود می‌آمد چون عدد ۱۰۰ به پارامتری از نوع رشته و رشته Hello World! به پارامتری از نوع اعداد صحیح ارسال می‌شد. این نشان می‌دهد که ترتیب ارسال آرگومانها به پارامترها هنگام فراخوانی متد مهم است. به مثال ۱ توجه کنید در آن مثال دو عدد از نوع int به پارامترها ارسال کردیم که ترتیب ارسال آنها چون هردو پارامتر از یک نوع بودند مهم نبود. ولی اگر پارامترهای متد دارای اهداف خاصی باشند ترتیب ارسال آرگومانها مهم است.

```
func ShowPersonStats(age, height int) {
    fmt.Printf("Age = %d \n", age)
    fmt.Printf("Height = %d", height);
}

//Using the proper order of arguments
ShowPersonStats(20, 160)

//Acceptable, but produces odd results
ShowPersonStats(160, 20)
```

در مثال بالا نشان داده شده است که، حتی اگر متد دو آرگومان با یک نوع داده‌ای قبول کند باز هم بهتر است ترتیب بر اساس تعریف پارامترها رعایت شود. به عنوان مثال در اولین فراخوانی متد بالا اشکالی به چشم نمی‌آید چون سن شخص ۲۰ و قد او ۱۶۰ سانتی متر است. اگر آرگومانها را به ترتیب ارسال نکنیم سن شخص ۱۶۰ و قد او ۲۰ سانتی متر می‌شود که به واقعیت نزدیک نیست. دانستن مبانی مقادیر برگشتی و ارسال آرگومانها باعث می‌شود که شما متدهای کارامد تری تعریف کنید. تکه کد زیر نشان می‌دهد که شما حتی می‌توانید مقدار برگشتی از یک متد را به عنوان آرگومان به متد دیگر ارسال کنید.

```
func MyMethod() int {
    return 5
}

func AnotherMethod(number int) {
    fmt.Println(number);
}

// Codes skipped for demonstration
AnotherMethod(MyMethod())
```

چون مقدار برگشتی متد MyMethod() عدد ۵ است و به عنوان آرگومان به متد AnotherMethod() ارسال می‌شود خروجی کد بالا هم عدد ۵ است.

ارسال آرایه به عنوان آرگومان

می‌توان آرایه‌ها را به عنوان آرگومان به متد ارسال کرد. ابتدا شما باید پارامترهای متد را طوری تعریف کنید، که آرایه دریافت کنند. به مثال زیر توجه کنید:

```
1  package main
2
3  import "fmt"
4
5  var i int
6  func TestArray(numbers [] int) {
7      for i=0; i < len(numbers); i++ {
8          fmt.Println(numbers[i]+1)
9      }
10 }
11
12 func main() {
13     var array = [] int { 1, 2, 3, 4, 5 }
14     TestArray(array)
15 }
```

```
2
3
4
5
6
```

مشاهده کردید که به سادگی می‌توان با تعریف یک آرایه در داخل پرانتزهای متد کاری کرد که آن متد، آرایه دریافت کند. وقتی متد در خط ۱۴ فراخوانی می‌شود، آرایه را فقط با استفاده از نام آن و بدون استفاده از اندیس ارسال می‌کنیم. برنامه بالا یک متد را نشان می‌دهد که یک آرایه را دریافت می‌کند و به هر یک از عناصر آن یک واحد اضافه می‌کند. در داخل متد ما مقادیر هر یک از اجزای آرایه را افزایش داده‌ایم. سپس از متد خارج شده و نتیجه را نشان می‌دهیم.

Variadic Functions

Variadic Functions امکان ارسال تعداد دلخواه پارامترهای همنوع و ذخیره آنها در یک آرایه ساده را فراهم می‌آورد. برای ایجاد متدی که به تعداد دلخواه پارامتر دریافت کند، از علامت سه نقطه (...) به صورت زیر استفاده می‌شود :

```
func methodName (vaiableName ... dataType) dataType{
   ...
}
```

همانطور که در کد بالا مشاهده می کنید، کافیست، آرگومان هایی که متد قرار است دریافت کن را به صورت vaiableName ... dataType بنویسید. یعنی یک نام، سپسس علامت ... و در نهایت نوع متغیرها را ذکر کنید. به مثال زیر توجه کنید:

```
1  package main
2
3  import "fmt"
4
5  func CalculateSum (numbers ... int) int {
6      var total int = 0
7
8      for _, num := range numbers {
9              total += num
10         }
11     return total
12 }
13
14 func main() {
15     fmt.Printf("1 + 2 + 3         = %d \n", CalculateSum(1, 2, 3))
16     fmt.Printf("1 + 2 + 3 + 4     = %d \n", CalculateSum(1, 2, 3, 4))
17     fmt.Printf("1 + 2 + 3 + 4 + 5 = %d \n", CalculateSum(1, 2, 3, 4, 5))
18 }
```

```
1 + 2 + 3         = 6
1 + 2 + 3 + 4     = 10
```

```
1 + 2 + 3 + 4 + 5 = 15
```

همانطور که در مثال بالا مشاهده می‌کنید، یک متد به نام CalculateSum() در خط ۵ تعریف شده است. برای اینکه این متد تعداد دلخواه پارامتر دریافت کند، از علامت سه نقطه (...) قبل از نوع داده‌ای پارامتر آن استفاده شده است. در اصل کلمه numbers یک آرایه است، که وقتی ما آرگومان‌ها را به متد ارسال می‌کنیم، در این آرایه ذخیره می‌شوند. حال متد را سه بار با تعداد مختلف آرگومانها فراخوانی می‌کنیم و سپس با استفاده از حلقه for این آرگومانها را جمع و به متد فراخوان برگشت می‌دهیم. وقتی از چندین پارامتر در یک متد استفاده می‌کنید، فقط یکی از آنها باید دارای علامت سه نقطه (...) بوده و همچنین از لحاظ مکانی باید آخرین پارامتر باشد. اگر این پارامتر (پارامتری که دارای سه نقطه است) در آخر پارامترهای دیگر قرار نگیرد و یا از چندین پارامتر سه نقطه دار استفاده کنید با خطا مواجه می‌شوید. به مثالهای اشتباه و درست زیر توجه کنید :

```go
func SomeFunction(varidic ... x, y int) //ERROR

func SomeFunction(x int, varidic ... y) //Correct
```

محدوده متغیر

متغیرها در Go دارای محدوده (scope) هستند. محدوده یک متغیر به شما می‌گوید که در کجای برنامه می‌توان از متغیر استفاده کرد و یا متغیر قابل دسترسی است. به عنوان مثال متغیری که در داخل یک متد تعریف می‌شود فقط در داخل بدنه متد قابل دسترسی است. می‌توان دو متغیر با نام یکسان در دو متد مختلف تعریف کرد. برنامه زیر این ادعا را اثبات می‌کند :

```go
package main

import "fmt"

func DemonstrateScope() {
    var number int = 5

    fmt.Printf("number inside method DemonstrateScope() = %d \n", number)
}

func main() {
    var number int = 10

    DemonstrateScope();

    fmt.Printf("number inside the Main method = %d", number)
}
```

```
number inside method DemonstrateScope() = 5
number inside the Main method = 10
```

مشاهده می‌کنید که حتی اگر ما دو متغیر با نام یکسان تعریف کنیم که دارای محدوده‌های متفاوتی هستند، می‌توان به هر کدام از آنها مقادیر مختلفی اختصاص داد. متغیر تعریف شده در داخل متد `()main` هیچ ارتباطی به متغیر داخل متد `()DemonstrateScope` ندارد. GO دارای دو محدوده است:

- **متغیرهای محلی**`(Local)`
- **متغیرهای سراسری**`(Global)`

متغیرهای محلی

متغیرهایی که داخل متد تعریف می‌شوند محلی هستند و فقط داخل همان متد قابل استفاده‌اند. به مثال زیر توجه کنید:

```go
package main

import "fmt"

func localVariable() {
    var number int = 10
    fmt.Println(number)
}

func main() {
    localVariable()
    fmt.Println(number)
}
```

همانطور که مشاهده می‌کنید با فراخوانی متد در خط ۱۱ مقدار متغیر number چاپ می‌شود ولی در خط ۱۲ که سعی در چاپ مقدار این متغیر داریم با پیغام خطا مواجه می‌شویم چون طول عمر این متغیر تا زمانی است که متد به پایان نرسیده است. با پایان متد متغیر و مقدار آن هم از بین می‌رود در نتیجه در خارج از متد نمی‌توان مقدار آن را چاپ کرد.

متغیرهای سراسری

متغیرهایی که در بیرون متد تعریف می‌شوند از نوع سراسری هستند. به مثال زیر توجه کنید:

```go
package main

import "fmt"

var number1 int = 10
var number2 int = 5
```

```
var sum     int

func globalVariable(){
    sum = number1 + number2
}

func main() {
    globalVariable()
    fmt.Println(sum)
}
```

```
15
```

همانطور که در کد بالا مشاهده می کنید، متغیرهایی که در خطوط ۶-۵ تعریف شده اند در داخل تمامی متدهای برنامه قابل دسترسی هستند. مثلا در خط ۱۰ مقدار دو متغیر number1 و number2 با هم جمع و در داخل متغیر sum قرار داده شده است. حال همین متغیر sum که در خارج از متد ()main تعریف شده است، قابل دسترسی و دارای مقدار ۱۵ می باشد.

بازگشت (Recursion)

بازگشت فرایندی است که در آن متد مدام خود را فراخوانی می‌کند تا زمانی که به یک مقدار مورد نظر برسد. بازگشت یک مبحث پیچیده در برنامه نویسی است و تسلط به آن کار راحتی نیست. به این نکته هم توجه کنید که بازگشت باید در یک نقطه متوقف شود وگرنه برای بی نهایت بار، متد، خود را فراخوانی می‌کند. در این درس یک مثال ساده از بازگشت را برای شما توضیح می‌دهیم. فاکتوریل یک عدد صحیح مثبت (n!) شامل حاصل ضرب همه اعداد مثبت صحیح کوچکتر یا مساوی آن می‌باشد. به فاکتوریل عدد ۵ توجه کنید.

```
5! = 5 * 4 * 3 * 2 * 1 = 120
```

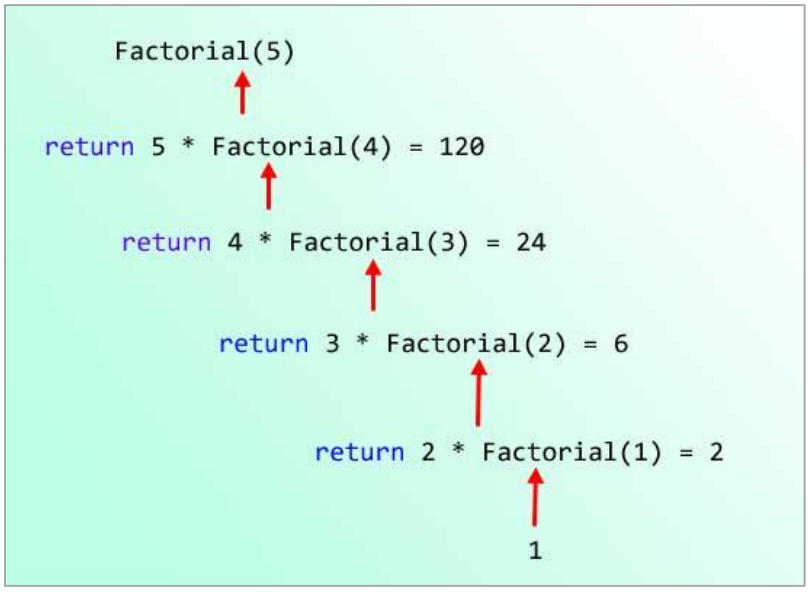
بنابراین برای ساخت یک متد بازگشتی باید به فکر توقف آن هم باشیم. بر اساس توضیح بازگشت، فاکتوریل فقط برای اعداد مثبت صحیح است. کوچک‌ترین عدد صحیح مثبت ۱ است. در نتیجه از این مقدار برای متوقف کردن بازگشت استفاده می‌کنیم.

```
package main

import "fmt"

func Factorial(number int) int {
    if (number == 1) {
        return 1
    }
    return number* Factorial(number - 1)
}

func main() {
```

```
13      fmt.Println(Factorial(5))
14  }
```

```
120
```

متد مقدار بزرگی را بر می‌گرداند چون محاسبه فاکتوریل می‌تواند خیلی بزرگ باشد. متد یک آرگومان که یک عدد است و می‌تواند در محاسبه مورد استفاده قرار گیرد را می‌پذیرد. در داخل متد یک دستور `if` می‌نویسیم و در خط `6` می گوییم که اگر آرگومان ارسال شده برابر ۱ باشد سپس مقدار ۱ را برگردان در غیر اینصورت به خط بعد برو. این شرط باعث توقف تکرارها نیز می‌شود. در خط ۹ مقدار جاری متغیر `number` در عددی یک واحد کمتر از خودش `(number - 1)` ضرب می‌شود. در این خط متد `Factorial` خود را فراخوانی می‌کند و آرگومان آن در این خط همان `number - 1` است. مثلاً اگر مقدار جاری `number 10` باشد یعنی اگر ما بخواهیم فاکتوریل عدد ۱۰ را به دست بیاوریم آرگومان متد `Factorial` در اولین ضرب ۹ خواهد بود. فرایند ضرب تا زمانی ادامه می‌یابد که آرگومان ارسال شده با عدد ۱ برابر نشود. شکل زیر فاکتوریل عدد ۵ را نشان می‌دهد.



از بازگشت در زمینه‌های خاصی در علوم کامپیوتر استفاده می‌شود. استفاده از بازگشت حافظه زیادی اشغال می‌کند پس اگر سرعت برای شما مهم است از آن استفاده نکنید.

ساختار (Struct)

ساختارها یا struct انواع داده‌ای هستند که توسط کاربر تعریف می‌شوند (user-define) و می‌توانند دارای فیلد و متد باشند. با ساختارها می‌توان نوع داده‌ای خیلی سفارشی ایجاد کرد. فرض کنید می‌خواهیم داده‌ای ایجاد کنیم که نه تنها نام شخص را ذخیره کند بلکه سن و حقوق ماهیانه او را نیز در خود جای دهد. برای تعریف یک ساختار به صورت زیر عمل می‌کنیم:

```
type StructName struct {
   member1
   member2
   member3
   ...
   member4
}
```

برای تعریف ساختار از کلمه کلیدی struct استفاده می‌شود. برای نامگذاری ساختارها از روش نامگذاری struct استفاده می‌شود. اعضا در مثال بالا (member1-5) می‌توانند متغیر باشند یا متد. در زیر مثالی از یک ساختار آمده است:

```
package main

import "fmt"

type Employee struct {
    name   string
    age    int
    salary float32
}

func main() {
    var employee1 Employee
    var employee2 Employee

    employee1.name   = "Jack"
    employee1.age    = 21
    employee1.salary = 1000

    employee2.name   = "Mark"
    employee2.age    = 23
    employee2.salary = 800

    fmt.Println("Employee 1 Details")
    fmt.Printf("Name  : %s    \n", employee1.name  )
    fmt.Printf("Age   : %d    \n", employee1.age   )
    fmt.Printf("Salary: $%.2f \n", employee1.salary)

    fmt.Println() //Seperator

    fmt.Println("Employee 2 Details")
    fmt.Printf("Name  : %s    \n", employee2.name  )
    fmt.Printf("Age   : %d    \n", employee2.age   )
    fmt.Printf("Salary: $%.2f \n", employee2.salary)
}
```

```
 Employee 1 Details
Name   : Jack
Age    : 21
Salary: $1000.00

Employee 2 Details
Name   : Mark
Age    : 23
Salary: $800.00
```

برای درک بهتر، کد بالا را شرح می‌دهیم. در خطوط ۹-۵ یک ساختار تعریف شده است. همانطور که در خط ۵ مشاهده می کنید ساختار با کلمه کلیدی `struct` تعریف می رود .نام ساختار نیز از روش نامگذاری پاسکال پیروی می‌کند. در داخل بدنه ساختار سه متغیر تعریف کرده‌ایم (۸-۶) ولی قبل از نام آنها از کلمه کلیدی `var` استفاده نکرده ایم (نباید هم این کار را بکنید). این سه متغیر مشخصات `Employee` (کارمند) مان را نشان می‌دهند. مثلاً یک کارمند دارای نام، سن و حقوق ماهانه می‌باشد. در خطوط ۱۲ و ۱۳ دو نمونه از ساختار `Employee` تعریف شده است. تعریف یک نمونه از ساختارها بسیار شبیه به تعریف یک متغیر معمولی است. ابتدا کلمه کلیدی`var` ، سپس یک نام و در آخر اسم ساختاری که ایجاد کرده ایم (`Employee`) را می نویسیم. در خطوط ۱۵ تا ۲۱ به متغیرهای مربوط به هر `Employee` مقادیری اختصاص می‌دهید. برای دسترسی به متغیرها در خارج از ساختار، ابتدا نام ساختار را تایپ کرده و سپس علامت دات (`.`) و در آخر نام متغیر را می‌نویسیم.

وقتی که از عملگر دات استفاده می‌کنیم این عملگر اجازه دسترسی به اعضای مخصوص آن ساختار را به شما می‌دهد. در خطوط ۲۴ تا ۲۶ نشان داده شده که شما چطور می‌توانید به مقادیر ذخیره شده در هر متغیر ساختار دسترسی یابید. می‌توان به ساختار، متد هم اضافه کرد. مثال زیر اصلاح شده مثال قبل است.

```go
package main

import "fmt"

type Employee struct {
    name   string
    age    int
    salary float32
}

func (e Employee) SayThanks() {
    fmt.Printf("%s thanked you! \n", e.name)
}

func main() {
    var employee1 Employee
    var employee2 Employee

```

```
    employee1.name   = "Jack"
    employee1.age    = 21
    employee1.salary = 1000

    employee2.name   = "Mark"
    employee2.age    = 23
    employee2.salary = 800

    fmt.Println("Employee 1 Details")
    fmt.Printf("Name  : %s    \n", employee1.name  )
    fmt.Printf("Age   : %d    \n", employee1.age   )
    fmt.Printf("Salary: $%.2f \n", employee1.salary)
    employee1.SayThanks()

    fmt.Println() //Seperator

    fmt.Println("Employee 2 Details")
    fmt.Printf("Name  : %s    \n", employee2.name  )
    fmt.Printf("Age   : %d    \n", employee2.age   )
    fmt.Printf("Salary: $%.2f \n", employee2.salary)
    employee1.SayThanks()
}
```

```
Employee 1 Details
Name: Jack
Age: 21
Salary: $1000.00
Jack thanked you!

Employee 2 Details
Name: Mike
Age: 23
Salary: $800.00
Mike thanked you!
```

در Go نمی توان یک متد در داخل ساختار ایجاد کرد. در خطوط ۱۱ تا ۱۳ یک متد در خارج ساختار تعریف شده است. این متد با استفاده از دستور (e Employee) بعد از کلمه کلیدی func به ساختار ارتباط داده شده است و مقدار فیلد name را گرفته و یک پیام منحصر به فرد برای هر نمونه نشان می‌دهد. برای فراخوانی متد، به جای اینکه بعد از علامت دات نام فیلد را بنویسیم، نام متد را نوشته و بعد از آن همانطور که در مثال بالا مشاهده می‌کنید (خطوط ۳۱ و ۳۹) پرانتزها را قرار می‌دهیم و در صورتی که متد به آرگومان هم نیاز داشت در داخل پرانتز آنها را می‌نویسیم.

رابط ها (Interfaces)

رابط‌ها انواعی هستند که فقط شامل تعاریفی برای متدها می‌باشند. رابط‌ها را می‌توان به عنوان پلاگین های ساختار در نظر گرفت. ساختاری که یک رابط خاص را پیاده سازی می‌کند لازم است که کدهایی برای اجرا توسط اعضا و متدهای آن فراهم کند چون اعضا و متدهای رابط هیچ کد اجرایی در بدنه خود ندارند. اجازه دهید که نحوه تعریف و استفاده از یک رابط در ساختار را توضیح دهیم:

```
1  package main
2
3  import "fmt"
4
5  type CA struct {
6      FullName  string
7      Age        int
8  }
9
10 type CB struct {
11     FirstName  string
12     LastName   string
13     PersonsAge int
14 }
15
16 func PrintInfo(item CA) {
17     fmt.Printf("Name: %s, Age %d \n", item.FullName, item.Age);
18 }
19
20 func main() {
21     a := CA{FullName: "John Doe", Age: 35}
22
23     PrintInfo(a)
24 }
```

در کد بالا دو ساختار CA و CB تعریف شده‌اند، در ساختار CA دو متغیر به نام FullName و Age و در ساختار CB سه متغیر به نام‌های FirstName و LastName ,PersonsAge تعریف کرده‌ایم. در برنامه بالا، یک متد به نام PrintInfo() داریم که یک پارامتر از نوع ساختار CA دارد. به شکل ساده در این متد مقدار متغیرهایشی ای که به این متد ارسال شده است، چاپ می‌شود. در متد main()یک شیء از ساختار CA ساخته‌ایم و متغیرهای آن را مقدار دهی کرده‌ایم (خط ۲۱). سپس این شیء را به متد PrintInfo ارسال می‌کنیم (خط ۲۳). ساختارهای CA و CB از نظر مفهومی شبیه یکدیگر هستند. مثلاً ساختار CA متغیر FullName را برای نمایش نام و نام خانوادگی دارد ولی ساختار CB برای نمایش نام و نام خانوادگی دو متغیر جدا از هم به نام‌های FirstName و LastName را دارد. و همچنین یک متغیر برای نگهداری مقدار سن داریم که در ساختار CA نام آن Age و در ساختار CB نام آن PersonAge می‌باشد. مشکل اینجاست که اگر ما یک نمونه از ساختار CA را به متد (PrintInfo) ارسال کنیم، از آنجایی که در داخل

بدنه این متد فقط مقدار دو متغیر چاپ می‌شود، اگر بخواهیم یک نمونه از ساختار CB را به آن ارسال کنیم که دارای سه متغیر است با خطا مواجه می‌شویم (زیرا متد PrintInfo() با ساختار ساختار CA سازگار است و متغیرهای CB را نمی‌شناسد). برای رفع این مشکل باید ساختار دو ساختار CA و CB را شبیه هم کنیم و این کار را با استفاده از Interface انجام می‌دهیم:

```
package main

import "fmt"

type IInfo interface {
    GetName() string
    GetAge()  int
}

type CA struct {
    FullName  string
    Age        int
}

type CB struct {
    FirstName  string
    LastName   string
    PersonsAge int
}

func (a CA) GetName() string {
    return a.FullName
}

func (a CA) GetAge() int {
    return a.Age
}

func (b CB) GetName() string {
    return b.FirstName + " " + b.LastName
}

func (b CB) GetAge() int {
    return b.PersonsAge
}

func PrintInfo(item IInfo) {
    fmt.Printf("Name: %s, Age %d \n", item.GetName(), item.GetAge());
}

func main() {
    a := CA {FullName: "John Doe", Age: 35}
    b := CB { FirstName: "Jane", LastName: "Rock", PersonsAge: 33}

    PrintInfo(a)
    PrintInfo(b)
}
```

```
Name: John Doe, Age 35
Name: Jane Rock, Age 33
```

کد بالا را می‌توان به اینصورت توضیح داد که، در خط ۸-۵ یک رابط به نام IInfo تعریف کرده‌ایم. برای پیاده سازی یک رابط در Go لازم نیست کار خاصی انجام دهیم. همین که متدهای آن را پیاده سازی و کدهای بدنه آنها را تکمیل کنیم، کفایت می‌کند. دو ساختار وظیفه دارند متدهای این رابط را پیاده سازی کنند، پس در خطوط ۲۷-۲۱ و ۳۹-۳۳ کدهای بدنه دو متد این رابط را آن طور که می‌خواهیم، می‌نویسیم. در خط ۳۷ متد PrintInfo() را طوری دستکاری می‌کنیم که یک پارامتر از نوع رابط دریافت کند. حال زمانی که دو نمونه از دو ساختار CA و CB در دو خط ۴۲ و ۴۳ ایجاد می‌کنیم و آنها را در دو خط ۴۵ و ۴۶ به متد PrintInfo() ارسال می‌کنیم، چونکه این دو ساختار متدهای رابط IInfo را پیاده سازی کرده‌اند، به طور صریح به رابط تبدیل می‌شود. یعنی ساختاری که یک رابط را پیاده سازی کند، به طور صریح می‌تواند به رابط تبدیل شود. حال بسته به اینکه شیء کدام ساختار به متد PrintInfo() ارسال شده است، متد مربوط به آن ساختار فراخوانی شده و مقادیر متغیرها چاپ می‌شود.

پایان ویرایش اول کتاب – ۹۷/۱۰/۱۷

راه‌های ارتباط با نویسنده

وب سایت: www.w3-farsi.com

لینک تلگرام: https://telegram.me/ebrahimi_younes

ID تلگرام: @ebrahimi_younes

پست الکترونیکی: younes.ebrahimi.1391@gmail.com