Some thoughts:

* Do not try to force yourself like “I have to learn it in X days”. The Beginner and Medium level are quite easy to reach but the Advanced level need a lot more time, so keep learning everyday. In my experience, it takes around 1-2 days to finish Beginner level, 3-7 days to pass Medium level and >= 2 weeks to comfortably working on a real product.
* Go is pretty simple so just start learning from web is enough. After that, feel free to read books to understand deeply in advanced things.
* Go is not C, Java… so please do not map the knowledge you already learned in other programming languages to Go. Just learn and do it the right (Go) way.
* The fastest way to learn programming is working on actual project(s), so grab an idea and code it in Go.
* Please be active and join other Go groups/events to get help and familiar with Go community.

I. Beginner

At this level, you should learn about Go basic syntaxes and its components. The key point of this level is you should learn slowly until becomes familiar with Go syntax and can freely write simple program (as noted at exercises).

What you should learn

* Primitive types: byte, int/uint (16, 32 and 64 bits), float (32⁄64 bits), bool, string, array and map.
  + Know how to cast from a type to another type.
* Variables and constants.
* If-else and switch-case.
* For loop.
* How to write a function.
* Handle errors.
* How to run/build Go program.

Documents

*Note: You don’t have to read the whole document at once, just read/practice the basics first.*

* Tour of Go (official document): <https://tour.golang.org/welcome/1>
* Go by example: <https://gobyexample.com/>

Exercises

1. Write a program to calculate the perimeter and square of a rectangle, triangle, round,…
2. Read the marks of Math, Physics and Chemistry from stdin, return the ranking based on average mark.
   * avgMark < 5: Bad
   * avgMark <8: Ok
   * avgMark < 9: Good
   * Otherwise Excellent

II. Medium

You should learn more about things listed in Beginner level and also learn new things.

What you should learn

* Package
* Init()
* Defer, panic, recover
* Slice
* Pointer
* Struct
* Interface
* Methods
* Concurrency primitives: goroutine, channel, select
* Go tools: gofmt, godoc, glide,…

Documents

* Continue reading the documents from Beginner level.
* Go tutorial series: <https://golangbot.com/learn-golang-series/>
* 50 shades of Go: <http://devs.cloudimmunity.com/gotchas-and-common-mistakes-in-go-golang/>
* Go Effective: <https://golang.org/doc/effective_go.html> (must read, but you can just read the parts you want to know first and come back later to read the whole document).

Exercises

1. Write TODO app.

III. Advanced

A lot of things you must learn in this level and actually we don’t have the time range since you must keep learning everyday. Basically, at this level, you must understand deeply on Go internals, concurrency programming and must work on Go projects to reinforce your knowledge.

What you should learn (deeply understand)

* Array and slice.
* Struct and interface.
* goroutine, channel and select.
* sync package (Mutex, RWMutex, WaitGroup, Pool…).
* Concurrency model:
  + Fan-in
  + Fan-out
  + Worker pool
  + Semaphore
  + …
* Databases
* net/http package: Write web server/services.
* Take a look on godoc of important Go’s packages.

Documents

* Go Wiki: <https://github.com/golang/go/wiki>
* Go Blog: <https://blog.golang.org/>
* Go source code: <https://github.com/golang/go>
* Go Academy: <https://www.youtube.com/channel/UCx9QVEApa5BKLw9r8cnOFEA>

Exercises

1. Write a crawler to crawl data from any website you like.
2. Write a web services that serve REST APIs.
3. Write your own blog.
4. Build a microservices system.

IV. More understand Go with Me

1. Why do you love Go?

=> The answer depends on yourself.

For me, I love Go by its simplicity and performance.

Other possible reasons: fast compiling, good tooling, cross-platform, native concurrency programming...

2. What is the differences between Go concurrency model vs other programming languages?

=> In other programming languages (i.e Java), threads typically has to "fight" each others in order to gain the access to resource (lock/unlock/synchronized).

Instead, Go uses Hoare's CSP (Communicating Sequential Processes) concurrency model, which requires communicating between goroutines (via channel) to access resource (the communication between goroutines is used as the synchronization method).

3. Tell me about Go concurrency?

=> Go concurrency achieved by using 3 components: goroutine, channel and select.

* <http://www.golangbootcamp.com/book/concurrency>

4. What is the differences between goroutine and thread?

=> Goroutine is basically the same as thread in other programming languages. Two main differences are:

  - Goroutine is spawn and scheduled by Go runtime while thread is managed by OS.

  - Goroutine is much more lightweight than thread (e.g.: smaller stack size, requires less instructions to schedule, manage than thread...)

You can spin hundreds of thousand goroutines in a single machine but it's (almost [?]) impossible to do the same with thread.

* <https://codeburst.io/why-goroutines-are-not-lightweight-threads-7c460c1f155f>

5. Go pass-by-value or pass-by-reference or both?

=> From higher view, you can both pass-by-value or pass-by-reference in Go. (acceptable answer)

But if you understand Go internal, actually there's no pass-by-reference in Go, everything is pass-by-value. (good answer)

* <https://dave.cheney.net/2017/04/29/there-is-no-pass-by-reference-in-go>

, size of slice can be changed at runtime.

Array is pass-by-value (copy) while slice is pass-by-reference.

* <https://blog.golang.org/go-slices-usage-and-internals>

7. Why Go has no “character” type but uses runes as the representation of “character”s in string?

=> Go is fully supported for Unicode encoding (UTF-8 impl).

In Unicode encoding, each “character” is represented as 1 code point. UTF-8 implementation uses from 1 to 4 bytes for each code point (depends on the position of Unicode code point).

So in Go, 1 “character” can be in size of 1, 2, 3, or even 4 bytes ⇒ Need a type to represent code point rather than fixed 1-byte-char likes other programming languages.

1 rune in Go is equal to 1 Unicode code point.

* <https://www.joelonsoftware.com/2003/10/08/the-absolute-minimum-every-software-developer-absolutely-positively-must-know-about-unicode-and-character-sets-no-excuses/>
* <https://blog.golang.org/strings>

9. Concurrency vs Parallelism:

Parallel: CPU có 2 cores, user vừa chơi game (trên 1 core), vừa đồng thời streaming game lên Twitch.tv (trên core thứ 2).

=> Tại 1 thời điểm, xử lý đc cùng lúc 2 việc trở lên => Parallel.

Concurrency: CPU chỉ có 1 core nhưng vẫn có thể mở được cùng lúc Word, Chrome, File Explorer...

=> Tại 1 thời điểm, CPU chỉ xử lý đc duy nhất 1 việc, nhưng OS vẫn có cách structure để run đc nhiều program mà ko bị blocking lẫn nhau.

=> Concurrency là cách design sao cho nhiều module của 1 program cùng run đc, không nhất thiết phải run đồng thời (parallel).

---------------------------

VD: Cho 1 mảng int từ 1-100, tính tổng.

=> Chia ra 2 goroutines, goroutine #1 tính tổng từ 1-50, goroutine #2 tính tổng từ 51-100.

=> Mỗi goroutine có thể coi như 1 module của program.

- TH1: Nếu program chạy trên CPU chỉ có 1 core: main -> goroutine #1 chạy -> goroutine #2 chạy -> Tổng.

- TH2: Nếu program chạy trên CPU có 2 cores trở lên:

        / -> goroutine #1 \

main                              --> Tổng

        \ -> goroutine #2 /

Việc cả 2 goroutines đc thực thi tại cùng 1 thời điểm như TH2 là parallelism.

Còn việc đưa ra solution để cho phép nhiều module (2 goroutines) có thể chạy độc lập vs nhau (không phân biệt chạy trên 1 CPU hay nhiều CPUs) như trên gọi là concurrency.

=> Parallelism là 1 trường hợp đặc biệt của concurrency.

1 program luôn có thể được xây dựng để chạy concurrency (by software), nhưng có đạt đc parallelism hay không phụ thuộc vào cả software (concurrency code) và hardware (CPU nhiều cores or nhiều CPU).