BASIC

1. How do you implement concurrency in Go?

Trong Go, concurrency được thực hiện chủ yếu thông qua goroutines và channels. Goroutine giống như một lightweight thread, rất dễ tạo bằng cách thêm go trước một hàm, ví dụ go myFunc(). Điều này cho phép nhiều công việc chạy đồng thời mà không tốn nhiều tài nguyên như thread thông thường. Để đồng bộ và giao tiếp giữa các goroutine, mình thường dùng channels – nó giống như một đường ống để gửi và nhận dữ liệu một cách an toàn giữa các goroutine. Ngoài ra, Go cũng có các cơ chế như select để quản lý nhiều channel cùng lúc, hoặc package sync nếu cần mutex hay WaitGroup. Nói chung, Go giúp viết code concurrent rất trực quan và an toàn.

"Concurrency trong Go thực hiện bằng goroutines và channels. Goroutine giống như thread nhẹ, dễ tạo với go myFunc(). Channels giúp giao tiếp an toàn giữa các goroutine, còn select và package sync hỗ trợ quản lý đồng thời nhiều tác vụ."

2. How do you handle errors in Go?

Trong Go, error không phải là exception như các ngôn ngữ khác, mà thường được trả về như một giá trị. Một hàm thường trả về value, err, và mình kiểm tra err ngay sau khi gọi hàm. Nếu err khác nil thì xử lý lỗi, ví dụ log, trả về lỗi lên caller, hoặc thực hiện fallback. Ngoài ra, Go có package errors để tạo hoặc wrap lỗi, giúp giữ thông tin gốc khi propagate. Mình cũng dùng defer kết hợp với recover chỉ trong những tình huống cực kỳ đặc biệt, vì Go khuyến khích xử lý lỗi trực tiếp qua giá trị trả về.

"Trong Go, lỗi được trả về như giá trị, thường là value, err. Mình kiểm tra err sau khi gọi hàm và xử lý ngay. Package errors giúp tạo hoặc wrap lỗi, còn defer và recover chỉ dùng trong tình huống đặc biệt."

3. How do you implement interfaces in Go?

Trong Go, interface định nghĩa một tập hợp các phương thức, và bất kỳ kiểu dữ liệu nào có triển khai đầy đủ các phương thức đó đều tự động thực thi interface, mà không cần khai báo kiểu implements như Java hay C#. Ví dụ, nếu mình có interface Reader với phương thức Read(p []byte) (n int, err error), thì bất kỳ struct nào triển khai phương thức Read với cùng chữ ký đều có thể được dùng như một Reader. Cách này rất linh hoạt và giúp code Go trở nên decoupled, dễ test và mở rộng.

"Trong Go, interface là tập hợp các phương thức, và một kiểu tự động thực hiện interface nếu triển khai đủ các phương thức. Không cần từ khóa implements, điều này giúp code linh hoạt và dễ mở rộng."

4. How do you optimize the performance of Go code?

Để tối ưu hiệu năng trong Go, trước tiên mình thường đo đạc bằng các công cụ profiling như pprof để biết đâu là điểm nghẽn. Sau đó, mình tập trung vào những việc như giảm allocations bằng cách tái sử dụng object hoặc slice, tránh tạo goroutine quá nhiều nếu không cần thiết, và tận dụng channel một cách hiệu quả. Ngoài ra, mình cũng chú ý đến algorithm và cấu trúc dữ liệu – vì chọn thuật toán tốt thường cải thiện hiệu năng nhiều hơn tối ưu nhỏ lẻ. Cuối cùng, nếu cần, mình dùng các kỹ thuật concurrency hợp lý, như worker pool, để tận dụng đa core mà không overload hệ thống.

"Mình tối ưu Go code bằng cách đo profile với pprof, giảm allocations, tái sử dụng object, chọn thuật toán và cấu trúc dữ liệu hợp lý, và quản lý goroutine hiệu quả với worker pool hoặc channel tối ưu."

5. What is the role of the "init" function in Go?

Trong Go, init là một hàm đặc biệt được gọi tự động khi package được load, trước hàm main. Mỗi package có thể có nhiều hàm init, và chúng thường được dùng để khởi tạo giá trị mặc định, thiết lập cấu hình, hoặc thực hiện các thao tác chuẩn bị trước khi chương trình chạy. Điểm quan trọng là bạn không gọi init trực tiếp; Go runtime sẽ tự động gọi nó theo thứ tự load package.

"Hàm init trong Go chạy tự động khi package được load, trước main. Nó thường dùng để khởi tạo giá trị, cấu hình hoặc chuẩn bị môi trường, và không được gọi trực tiếp."

6. What are dynamic and static types of declaration of a variable in Go?

Trong Go, kiểu dữ liệu của biến có thể được khai báo theo hai cách: tĩnh (static) hoặc động (dynamic). Với khai báo tĩnh, bạn chỉ định rõ kiểu khi khai báo, ví dụ var x int = 10; kiểu này được xác định lúc biên dịch và không thể thay đổi. Còn khai báo động, Go sẽ tự suy luận kiểu từ giá trị gán, ví dụ x := 10; ở đây Go sẽ hiểu x là int dựa trên giá trị 10. Tóm lại, static type giúp rõ ràng và an toàn, còn dynamic type giúp code ngắn gọn hơn nhờ type inference.

"Trong Go, static type là khi khai báo rõ kiểu, ví dụ var x int = 10, còn dynamic type là khi Go tự suy luận kiểu từ giá trị, ví dụ x := 10. Static type an toàn hơn, dynamic type ngắn gọn hơn."

7. What is the syntax for declaring a variable in Go?

Trong Go, có vài cách để khai báo biến. Cách truyền thống là dùng từ khóa var kèm tên biến và kiểu, ví dụ var x int = 10. Go cũng cho phép khai báo mà không cần gán giá trị ngay, ví dụ var y string, khi đó biến sẽ có giá trị mặc định kiểu đó. Một cách khác là khai báo ngắn gọn với :=, ví dụ z := 20; ở đây Go tự suy luận kiểu của biến từ giá trị. Cả hai cách đều phổ biến và tùy vào ngữ cảnh bạn muốn code rõ ràng hay ngắn gọn.

"Trong Go, khai báo biến có thể dùng var x int = 10 hoặc x := 10. var rõ ràng kiểu, := để Go tự suy luận kiểu từ giá trị."

8. What are Golang packages?

Trong Go, package là một cách tổ chức code thành các module hoặc thư viện riêng biệt. Mỗi package chứa các file .go và định nghĩa các hàm, struct, interface, hay biến có liên quan với nhau. Khi mình muốn dùng code từ package khác, mình chỉ cần import package đó. Package giúp code trở nên modular, dễ quản lý, tái sử dụng và test. Ví dụ package fmt chứa các hàm để format input/output như Println.

"Package trong Go là cách tổ chức code thành module riêng, chứa hàm, struct hay biến liên quan. Mình dùng import để dùng code từ package khác. Nó giúp code modular, dễ quản lý và tái sử dụng."

9. What are the different types of data types in Go?

Trong Go, có nhiều loại kiểu dữ liệu cơ bản và phức tạp. Kiểu cơ bản gồm số nguyên (int, int8, int16, int32, int64), số thực (float32, float64), số phức (complex64, complex128), boolean (bool) và chuỗi (string). Bên cạnh đó, Go có các kiểu composite như array, slice, map, struct và pointer. Ngoài ra, Go cũng hỗ trợ interface và function như kiểu dữ liệu. Kiểu dữ liệu trong Go rất rõ ràng và mạnh mẽ, giúp kiểm tra lỗi ngay khi biên dịch.

"Go có kiểu cơ bản như int, float, bool, string, complex; và kiểu composite như array, slice, map, struct, pointer, interface và function. Kiểu dữ liệu rõ ràng giúp code an toàn và dễ maintain."

10. How do you create a constant in Go?

Trong Go, để tạo một hằng số mình dùng từ khóa const. Ví dụ, const Pi = 3.14 sẽ tạo một hằng số Pi kiểu float64 mặc định. Hằng số là giá trị không thay đổi trong suốt chương trình, và Go hỗ trợ các hằng số số học, boolean, hoặc string. Mình cũng có thể khai báo nhiều hằng số cùng lúc bằng cú pháp block, ví dụ: const (A = 1; B = 2). Hằng số giúp code rõ ràng và tránh lỗi do giá trị bị thay đổi vô ý.

"Trong Go, mình dùng const để tạo hằng số, ví dụ const Pi = 3.14. Hằng số không thay đổi, có thể là số, boolean hoặc string, và giúp code an toàn và rõ ràng."

11. What data types does Golang use?

Golang có nhiều loại kiểu dữ liệu. Kiểu cơ bản gồm số nguyên (int, int8, int16, int32, int64), số thực (float32, float64), số phức (complex64, complex128), boolean (bool) và chuỗi (string). Ngoài ra, Go có các kiểu composite như array, slice, map, struct, pointer, và function. Go cũng có interface như một kiểu dữ liệu trừu tượng. Những kiểu này giúp Go mạnh mẽ trong việc kiểm tra lỗi khi biên dịch và viết code rõ ràng, an toàn.

"Golang có kiểu cơ bản như int, float, bool, string, complex; và kiểu composite như array, slice, map, struct, pointer, function, cùng interface. Kiểu dữ liệu rõ ràng giúp code an toàn và dễ maintain."

12. Distinguish unbuffered from buffered channels.

Trong Go, channel là cơ chế để giao tiếp giữa các goroutine. Một unbuffered channel không có bộ đệm, nghĩa là khi một goroutine gửi dữ liệu, nó sẽ block cho đến khi goroutine khác nhận dữ liệu. Ngược lại, một buffered channel có thể lưu trữ một số lượng phần tử nhất định, ví dụ ch := make(chan int, 5). Goroutine gửi dữ liệu vào buffered channel sẽ chỉ block khi buffer đầy, còn nhận dữ liệu sẽ block khi buffer trống. Như vậy, buffered channels cho phép một số độ trễ và linh hoạt trong đồng bộ, trong khi unbuffered channels yêu cầu đồng bộ trực tiếp giữa sender và receiver.

"Unbuffered channel block khi gửi dữ liệu cho đến khi có goroutine nhận, trong khi buffered channel có bộ đệm, gửi chỉ block khi buffer đầy và nhận block khi buffer trống. Buffered channels cho phép đồng bộ linh hoạt hơn."

13. Explain string literals.

Trong Go, string literal là cách viết trực tiếp giá trị chuỗi trong code. Có hai loại string literal chính: interpreted string literal dùng dấu nháy kép "...", trong đó các ký tự escape như \n, \t sẽ được xử lý, và raw string literal dùng backtick `...`, giữ nguyên tất cả ký tự, bao gồm xuống dòng và khoảng trắng. String literals giúp mình biểu diễn chuỗi cố định trực tiếp trong chương trình, thuận tiện cho text, template, hoặc biểu thức regex.

"String literal trong Go là giá trị chuỗi viết trực tiếp. "..." là interpreted string với escape characters, `...` là raw string giữ nguyên mọi ký tự, xuống dòng và khoảng trắng."

14. What is a Goroutine and how do you stop it?

Trong Go, goroutine là một lightweight thread, cho phép chạy nhiều công việc đồng thời rất hiệu quả về bộ nhớ. Bạn tạo goroutine bằng cách thêm go trước một hàm, ví dụ go myFunc(). Goroutine không có cách trực tiếp để 'kill' như thread trong một số ngôn ngữ khác. Thay vào đó, mình thường dùng cancellation pattern, ví dụ context với context.WithCancel, hoặc channel để báo cho goroutine tự dừng. Goroutine sẽ check context hoặc channel và tự thoát một cách an toàn. Đây là cách Go khuyến khích để tránh việc dừng thread một cách thô bạo.

"Goroutine là lightweight thread trong Go, tạo bằng go myFunc(). Không thể dừng trực tiếp; thường dùng context hoặc channel để báo goroutine tự thoát an toàn."

15. What is the syntax for creating a function in Go?

Trong Go, để tạo một hàm mình dùng từ khóa func, theo cú pháp func functionName(parameters) returnType { /* body */ }. Ví dụ, func add(a int, b int) int { return a + b }. Nếu hàm không trả về giá trị thì chỉ bỏ return type, ví dụ func sayHello(name string) { fmt.Println("Hello", name) }. Go cũng hỗ trợ trả về nhiều giá trị, ví dụ func swap(a, b int) (int, int) { return b, a }. Cách viết này rất rõ ràng và giúp code dễ đọc.

"Tạo hàm trong Go dùng func name(params) returnType { body }. Hàm có thể không trả về gì hoặc trả về nhiều giá trị. Ví dụ func add(a, b int) int { return a + b }."

16. How do you create a loop in Go?

Trong Go, loop chỉ có một từ khóa là for, nhưng nó rất linh hoạt. Cú pháp phổ biến là for init; condition; post { /* body */ }, ví dụ for i := 0; i < 5; i++ { fmt.Println(i) }. Ngoài ra, for cũng có thể dùng như while loop: for condition { /* body */ }. Hoặc dùng for không điều kiện để tạo infinite loop: for { /* body */ }. Go không có while hay do-while; tất cả đều dùng for.

"Go dùng for cho tất cả vòng lặp. Có thể viết for i := 0; i < n; i++, for condition, hoặc for {} cho infinite loop. Go không có while hay do-while."

17. What is the syntax for an if statement in Go?

Trong Go, cú pháp của if khá trực quan. Ví dụ cơ bản là if condition { /* body */ }. Go cũng hỗ trợ else và else if, ví dụ if x > 0 { ... } else if x < 0 { ... } else { ... }. Một điểm đặc biệt là Go cho phép khai báo biến ngay trong câu lệnh if, ví dụ if v := compute(); v > 10 { ... }. Biến này chỉ tồn tại trong scope của if-else block.

"Cú pháp if trong Go là if condition { }, có thể dùng else if và else. Có thể khai báo biến ngay trong câu lệnh, ví dụ if v := compute(); v > 10 { }."

18. What are some benefits of using Go?

Go có nhiều lợi ích. Đầu tiên, nó nhanh và hiệu quả, cả về tốc độ biên dịch lẫn runtime, nhờ hệ thống garbage collector nhẹ và compile trực tiếp ra machine code. Thứ hai, Go có concurrency mạnh mẽ với goroutines và channels, giúp viết code đa luồng dễ dàng. Thứ ba, Go có cú pháp đơn giản và rõ ràng, dễ đọc, dễ maintain, phù hợp cho cả team lớn. Ngoài ra, Go có bộ công cụ chuẩn mạnh mẽ, hỗ trợ testing, formatting, profiling sẵn, và cross-platform tốt, nên viết ứng dụng server hoặc network rất tiện.

"Go nhanh, hiệu quả, có concurrency mạnh với goroutines, cú pháp đơn giản, dễ maintain, và bộ công cụ chuẩn mạnh mẽ cho testing, profiling, và cross-platform."

19. How do you create a pointer in Go?

Trong Go, pointer là biến lưu trữ địa chỉ của một giá trị khác. Để tạo pointer, mình dùng toán tử & để lấy địa chỉ của biến, ví dụ p := &x. Để truy cập hoặc thay đổi giá trị mà pointer trỏ tới, mình dùng toán tử *, ví dụ *p = 10. Go không hỗ trợ pointer arithmetic như C, nên an toàn hơn, nhưng vẫn giúp chia sẻ dữ liệu hoặc cập nhật trực tiếp giá trị trong hàm.

*"Pointer trong Go lưu địa chỉ của biến. Dùng p := &x để tạo pointer, và *p để truy cập hoặc thay đổi giá trị mà nó trỏ tới. Go không có pointer arithmetic nên an toàn."*

20. What is the syntax for creating a struct in Go?

Trong Go, struct là kiểu dữ liệu composite dùng để nhóm nhiều giá trị có liên quan lại với nhau. Cú pháp tạo struct là dùng từ khóa type để định nghĩa tên struct. Struct giúp tổ chức dữ liệu rõ ràng và kết hợp với phương thức để mô tả hành vi của đối tượng.

"Struct trong Go nhóm nhiều giá trị lại với nhau. Dùng type Name struct { fields } để định nghĩa, và p := Name{Field1: val1} để tạo instance."

21. How do you create an array in Go?

Trong Go, array là kiểu dữ liệu chứa một số lượng phần tử cố định cùng kiểu. Cú pháp tạo array là var arr [size]Type, ví dụ var numbers [5]int sẽ tạo mảng 5 phần tử kiểu int với giá trị mặc định là 0. Mình cũng có thể khởi tạo với giá trị cụ thể: arr := [3]int{1, 2, 3}. Go còn hỗ trợ khai báo array tự động xác định kích thước bằng ..., ví dụ arr := [...]int{1,2,3}.

"Array trong Go là tập hợp số lượng phần tử cố định cùng kiểu. Dùng var arr [size]Type hoặc arr := [3]int{1,2,3}. Có thể dùng ... để Go tự xác định kích thước."

22. How will you perform inheritance with Golang?

Go không có inheritance theo kiểu class-based như Java hay C++. Thay vào đó, Go dùng composition để tái sử dụng code.

"Go không hỗ trợ inheritance trực tiếp, mà dùng composition bằng cách nhúng struct và interface. Struct nhúng sẽ thừa kế các phương thức, interface giúp polymorphism."

23. How do you create a slice in Go?

Trong Go, slice là một kiểu dữ liệu linh hoạt, đại diện cho một phần của array, có độ dài có thể thay đổi. Có nhiều cách tạo slice:

Dùng cú pháp literal: s := []int{1, 2, 3}.

Dùng make: s := make([]int, 5) tạo slice 5 phần tử kiểu int với giá trị mặc định 0.

Lấy một phần của array hoặc slice khác: s2 := s[1:3].

Slices rất phổ biến vì linh hoạt hơn array, không cần kích thước cố định và có thể mở rộng.

"Slice trong Go là array linh hoạt, độ dài có thể thay đổi. Tạo bằng literal s := []int{1,2,3}, make([]int, 5), hoặc cắt từ array/slice khác s2 := s[1:3]."

24. What is the difference between an array and a slice in Go?

Trong Go, array có kích thước cố định và kiểu dữ liệu xác định, ví dụ var a [5]int. Array lưu trữ tất cả phần tử liền nhau trong bộ nhớ và kích thước không thể thay đổi sau khi khai báo. Ngược lại, slice là wrapper linh hoạt trên array, có thể thay đổi độ dài, ví dụ s := []int{1,2,3} hoặc s = append(s, 4). Slice lưu trữ con trỏ tới array gốc, chiều dài và capacity, nên thao tác với slice nhẹ hơn và tiện lợi hơn. Nói chung, slice được sử dụng phổ biến hơn array trong Go vì linh hoạt.

"Array có kích thước cố định, slice là wrapper linh hoạt trên array, có thể thay đổi độ dài và dùng append. Slice nhẹ hơn và phổ biến hơn array."

25. How do you create a map in Go?

Trong Go, map là kiểu dữ liệu key-value, rất giống dictionary trong Python hoặc object trong JavaScript. Để tạo map, có thể dùng literal hoặc dùng make. Map trong Go rất linh hoạt, key phải là kiểu có thể so sánh (comparable), còn value có thể là bất kỳ kiểu nào. Map thường dùng để lưu dữ liệu tra cứu nhanh.

"Map trong Go là kiểu key-value. Tạo bằng literal m := map[string]int{"a":1} hoặc m := make(map[string]int) và gán giá trị. Key phải comparable, value có thể bất kỳ kiểu nào."

26. How do you iterate through a map in Go?

Trong Go, để lặp qua map mình dùng for range.Cú pháp for key, value := range map trả về key và value từng cặp. Nếu chỉ muốn key hoặc value, có thể bỏ phần không dùng bằng _. Map trong Go không đảm bảo thứ tự, nên mỗi lần lặp thứ tự key có thể khác nhau.
"Dùng for key, value := range map để lặp qua map. Có thể bỏ key hoặc value bằng _. Thứ tự key không cố định."

27. What is a goroutine in Go?

Trong Go, goroutine là một lightweight thread, dùng để chạy code đồng thời. Bạn tạo goroutine bằng cách thêm go trước một hàm, ví dụ go myFunc(). Goroutine rất nhẹ, nhiều goroutine có thể chạy cùng lúc trên một thread OS, giúp tận dụng concurrency hiệu quả. Go runtime quản lý scheduling và stack của goroutine, vì vậy chúng rất tiết kiệm bộ nhớ so với thread truyền thống.
"Goroutine là lightweight thread trong Go, chạy code đồng thời. Tạo bằng go myFunc(). Rất nhẹ, nhiều goroutine có thể chạy cùng lúc."

28. What are the looping constructs in Go?

Trong Go, loop chỉ có một từ khóa là for, nhưng rất linh hoạt. Bạn có thể viết loop theo nhiều cách:
Cú pháp truyền thống: for init; condition; post { ... }
Loop kiểu while: for condition { ... }
Infinite loop: for { ... }
Go không có while hay do-while; tất cả vòng lặp đều dùng for. Ngoài ra, for có thể kết hợp với range để lặp qua array, slice, string, map hoặc channel.
"Go chỉ có for cho tất cả vòng lặp: truyền thống (for init; cond; post), kiểu while (for cond), infinite loop (for {}), và dùng for range để lặp qua array, slice, map, string hoặc channel."

29. What is a channel in Go?

Trong Go, channel là cơ chế giao tiếp giữa các goroutine, giúp gửi và nhận dữ liệu một cách an toàn mà không cần dùng mutex. Bạn có thể tạo channel bằng ch := make(chan int). Goroutine có thể gửi dữ liệu vào channel bằng ch <- value và nhận dữ liệu bằng value := <-ch. Go hỗ trợ unbuffered channel, nơi gửi sẽ block cho đến khi có goroutine nhận, và buffered channel, nơi gửi block chỉ khi buffer đầy. Channels giúp quản lý concurrency dễ dàng và an toàn hơn.
"Channel là cơ chế giao tiếp giữa goroutine trong Go. Tạo bằng make(chan Type), gửi bằng ch <- value, nhận bằng value := <-ch. Unbuffered block khi gửi, buffered block khi đầy."

30. How do you create a channel in Go?

Trong Go, để tạo một channel mình dùng make. Ví dụ, để tạo unbuffered channel kiểu int: ch := make(chan int). Nếu muốn tạo buffered channel với capacity 5: ch := make(chan int, 5). Sau đó goroutine có thể gửi dữ liệu vào channel bằng ch <- value và nhận bằng value := <-ch. Channels giúp truyền dữ liệu giữa goroutine an toàn mà không cần mutex.
"Tạo channel trong Go dùng make(chan Type) cho unbuffered hoặc make(chan Type, capacity) cho buffered. Dùng <- để gửi và nhận giá trị."

31. How do you close a channel in Go?

Trong Go, để đóng một channel, mình dùng hàm built-in close(), ví dụ close(ch). Việc đóng channel báo cho các goroutine nhận biết rằng không còn dữ liệu nào được gửi vào nữa. Lưu ý là chỉ goroutine gửi dữ liệu mới nên đóng channel; không bao giờ đóng channel từ phía receiver vì sẽ gây panic nếu có goroutine khác đang gửi. Khi nhận dữ liệu từ channel đã đóng, sẽ nhận giá trị zero của kiểu dữ liệu.

"Dùng close(ch) để đóng channel. Chỉ nên đóng từ phía sender. Receiver sẽ nhận giá trị zero khi channel đã đóng."

32. How do you range over a channel in Go?

Trong Go, để lặp qua tất cả các giá trị từ một channel, mình dùng for range. Cú pháp for v := range ch sẽ tự động dừng khi channel đã đóng. Đây là cách an toàn để nhận tất cả dữ liệu từ channel mà không cần kiểm tra manually.

"Dùng for v := range ch để lặp qua channel. Vòng lặp tự dừng khi channel đã đóng."

INTERMEDIATE

1. How do you handle panics and recover from them in Go?

Trong Go, panic xảy ra khi chương trình gặp lỗi nghiêm trọng, như truy cập index ngoài mảng hoặc nil pointer. Để xử lý và tránh chương trình crash, Go có cơ chế recover, thường dùng defer để gọi một hàm bắt lỗi. Bất kỳ panic nào xảy ra trong hàm sẽ được catch bởi recover trong defer. Đây là cách Go xử lý lỗi nghiêm trọng an toàn mà vẫn cho phép tiếp tục chương trình.

"Dùng defer kết hợp recover() để bắt panic. Ví dụ, trong defer kiểm tra if r := recover(); r != nil { ... }. Giúp chương trình không crash khi gặp lỗi nghiêm trọng."

2. Explain the defer statement in Golang. Give an example of a deferred function's call.

Trong Go, defer dùng để trì hoãn việc thực hiện một hàm cho đến khi hàm chứa nó kết thúc. Điều này rất hữu ích để giải phóng tài nguyên, đóng file, hoặc unlock mutex mà không cần viết ở nhiều chỗ.

Nếu có nhiều defer, chúng sẽ được thực hiện theo ngăn xếp LIFO, nghĩa là defer gọi sau sẽ thực hiện trước.

"defer trì hoãn gọi một hàm cho đến khi hàm chứa nó kết thúc, hữu ích để giải phóng tài nguyên. Các defer được gọi theo thứ tự LIFO."

3. How do you create and use a package in Go?

Trong Go, package là cách tổ chức code thành module hoặc thư viện. Để tạo package, bạn tạo một thư mục và thêm file Go với dòng đầu tiên là package <name>.

Sau đó, trong file khác hoặc package main, bạn import package đó bằng import "<module-path>/<package name>"

Packages giúp code modular, dễ quản lý, tái sử dụng và test.

"Tạo package bằng package <name> trong file Go. Import package bằng import "<path>" và gọi các hàm bằng package.Func(). Packages giúp code modular và tái sử dụng."

4. What is the difference between a package and a module in Go?

Trong Go, package là tập hợp các file .go cùng thuộc một namespace, chứa các hàm, struct, interface hoặc biến liên quan. Nó giúp tổ chức code trong phạm vi nhỏ, dễ import và sử dụng trong chương trình. Ví dụ package fmt chứa các hàm format I/O.
Ngược lại, module là một tập hợp các package, được quản lý bằng Go Modules với file go.mod. Module định nghĩa versioning và dependency, ví dụ github.com/user/project có thể chứa nhiều package. Nói cách khác, package là đơn vị tổ chức code, còn module là đơn vị quản lý dependency và version.
"Package là tập hợp file Go cùng namespace, chứa hàm, struct, biến; module là tập hợp các package được quản lý bởi Go Modules với version và dependency."

5. How do you create a custom type in Go?

Trong Go, bạn có thể tạo custom type dựa trên kiểu dữ liệu hiện có bằng từ khóa type. Custom type rất hữu ích khi muốn tạo code rõ ràng hơn hoặc định nghĩa các phương thức cho kiểu mới. Cũng có thể tạo struct hoặc interface mới bằng cách tương tự.
"Dùng từ khóa type để tạo custom type từ kiểu có sẵn, ví dụ type Age int. Có thể tạo struct hoặc interface mới theo cách tương tự."

6. What is the syntax for type casting in Go?

Trong Go, type casting (hoặc type conversion) dùng cú pháp T(v), trong đó T là kiểu muốn chuyển sang và v là giá trị hiện tại.
Go không tự động chuyển kiểu như một số ngôn ngữ khác, nên type conversion phải rõ ràng. Điều này giúp tránh lỗi bất ngờ và làm code an toàn hơn.
"Dùng cú pháp T(v) để convert kiểu trong Go. Ví dụ float64(i) để chuyển int sang float64. Go không tự động convert, phải làm rõ ràng."

7. How do you use the "blank identifier" in Go?

Trong Go, blank identifier _ dùng để bỏ qua giá trị không cần sử dụng.
Nó cũng được dùng để import package mà không trực tiếp sử dụng.
Blank identifier giúp tránh lỗi biên dịch về unused variable và giữ code sạch sẽ.
"_ dùng để bỏ qua giá trị không cần dùng, ví dụ khi chỉ lấy một phần giá trị trả về từ hàm hoặc import package mà không dùng trực tiếp."

8. How do you create and use a pointer to a struct in Go?

Trong Go, để tạo pointer tới một struct, bạn dùng toán tử & để lấy địa chỉ của struct.
Go cho phép dùng ptr.Field thay vì phải viết (*ptr).Field, giúp code gọn hơn. Pointer tới struct hữu ích khi muốn truyền struct vào hàm mà không copy toàn bộ dữ liệu.
"Dùng & để tạo pointer tới struct, ví dụ ptr := &p. Có thể truy cập và sửa field bằng ptr.Field. Giúp truyền struct vào hàm mà không copy dữ liệu."

9. How do you embed a struct in Go?

Trong Go, struct embedding là cách tái sử dụng code và thực hiện ‘composition’. Nhúng một struct vào struct khác bằng cách khai báo struct đó mà không cần tên field.
Đây là cách Go thay thế inheritance truyền thống bằng composition.
"Embed struct bằng cách khai báo struct khác bên trong struct mà không đặt tên field. Struct mới sẽ có tất cả field và phương thức của struct nhúng."

10. How do you create and use a function closure in Go?

Trong Go, closure là một hàm mà có thể truy cập các biến từ phạm vi bên ngoài nó. Bạn có thể tạo closure bằng cách khai báo một hàm bên trong hàm khác hoặc gán hàm cho biến.
Closures hữu ích khi muốn giữ trạng thái hoặc tạo các hàm tùy chỉnh.
"Closure là hàm có thể truy cập biến từ scope bên ngoài. Ví dụ, add5 := makeAdder(5) trả về hàm func(int) int dùng biến x từ ngoài hàm."

11. What is the syntax for creating and using a function literal in Go?

Trong Go, function literal là một hàm không có tên, thường được gán cho biến hoặc dùng trực tiếp. Cũng có thể gọi ngay function literal mà không gán.
Function literal rất hữu ích khi muốn tạo closure, callback, hoặc truyền hàm vào các hàm khác.
"Function literal là hàm không tên. Tạo bằng f := func(params) { ... } và dùng f(args) hoặc gọi ngay func(params){...}(args)."

12. How do you use the "select" statement in Go?

Trong Go, select là cách để làm việc với nhiều channel đồng thời, tương tự như switch nhưng dành cho channel.
select sẽ chờ cho một trong các case có thể thực hiện, rồi thực thi case đó. Nếu nhiều case cùng khả thi, Go chọn ngẫu nhiên một case. default là case tùy chọn thực hiện nếu không có case nào sẵn sàng. Đây là cách rất mạnh để xử lý concurrency và tránh block.
"Use select to work with multiple channels. Each case handles send or receive. It executes a ready case, randomly if multiple are ready. Optional default runs if none are ready."

13. What is the syntax for creating and using a type assertion in Go?

Trong Go, type assertion dùng để chuyển một interface sang kiểu cụ thể mà interface đó chứa. Cú pháp là value.(Type)
Nếu không chắc chắn interface chứa kiểu đó, bạn có thể dùng form hai giá trị.
Type assertion rất hữu ích khi làm việc với interface và cần thao tác kiểu dữ liệu thực sự.
"Dùng value.(Type) để type assertion. Nếu không chắc, dùng v, ok := value.(Type) để tránh panic khi kiểu không khớp."

14. What is the syntax for creating and using a type switch in Go?

Trong Go, type switch dùng để kiểm tra kiểu thực sự của một interface. Cú pháp giống switch nhưng dùng .(type) trong case.
Type switch rất hữu ích khi muốn xử lý nhiều kiểu dữ liệu khác nhau mà interface có thể chứa. v trong mỗi case sẽ có kiểu tương ứng với case đó.
"Type switch kiểm tra kiểu của interface, dùng switch v := i.(type). Các case sẽ xử lý kiểu cụ thể, v có kiểu tương ứng."

15. What is the syntax for creating and using a type conversion in Go?

Trong Go, type conversion dùng để chuyển một giá trị từ kiểu này sang kiểu khác, và cú pháp rất đơn giản: T(v), trong đó T là kiểu muốn chuyển sang và v là giá trị hiện tại.
Go không tự động chuyển kiểu như một số ngôn ngữ khác, nên type conversion phải rõ ràng để tránh lỗi bất ngờ và giữ code an toàn.
"Dùng cú pháp T(v) để chuyển kiểu trong Go, ví dụ float64(i) để chuyển int sang float64. Go không tự động convert, phải làm rõ ràng."

16. How do you use the "sync" package to protect shared data in Go?

Trong Go, khi nhiều goroutine truy cập dữ liệu chung, mình cần tránh race condition. Gói sync cung cấp Mutex để bảo vệ dữ liệu.
Mỗi lần một goroutine muốn thay đổi giá trị 1 biến, nó phải Lock trước và Unlock sau, đảm bảo chỉ một goroutine truy cập tại một thời điểm. Ngoài ra, sync còn có RWMutex để tối ưu khi có nhiều reader và ít writer, và các cơ chế khác như WaitGroup để đồng bộ goroutine.
"Dùng sync.Mutex để bảo vệ shared data. Goroutine Lock trước khi thay đổi dữ liệu và Unlock sau khi xong, tránh race condition. Có thể dùng RWMutex nếu nhiều reader."

17. How do you use the "sync/atomic" package to perform atomic operations in Go?

Trong Go, gói sync/atomic cung cấp các hàm để thực hiện thao tác nguyên tử trên biến, giúp tránh race condition mà không cần dùng mutex.
Các thao tác này đảm bảo rằng khi nhiều goroutine cùng thay đổi biến, không xảy ra race condition. Đây là cách hiệu quả khi muốn cập nhật biến đơn giản mà không overhead của mutex.
"Dùng sync/atomic để thực hiện thao tác nguyên tử trên biến. Ví dụ atomic.AddInt32(&counter, 1) tăng biến mà không cần mutex, atomic.LoadInt32 để đọc, atomic.StoreInt32 để gán."

18. How do you use the "context" package to carry around request-scoped values in Go?

Trong Go, package context dùng để truyền dữ liệu, timeout, hoặc tín hiệu hủy giữa các goroutine trong phạm vi request. Context giúp bạn tránh dùng biến toàn cục, đồng thời quản lý timeout hoặc hủy request một cách gọn gàng khi cần.
"Dùng context để truyền request-scoped values. Tạo context gốc, thêm giá trị bằng context.WithValue, và truy xuất trong hàm bằng ctx.Value(key)."

19. How do you use the "net/http" package to build an HTTP server in Go?

Trong Go, net/http là package chuẩn để xây dựng HTTP server. Cách đơn giản nhất là dùng http.HandleFunc để đăng ký route và handler, sau đó gọi http.ListenAndServe để khởi chạy server.

net/http cũng hỗ trợ middleware, router phức tạp, và xử lý request/response dễ dàng.

"Dùng http.HandleFunc(path, handler) để đăng ký route và http.ListenAndServe(":8080", nil) để chạy server. Handler nhận ResponseWriter và Request."

20. How do you use the "encoding/json" package to parse and generate JSON in Go?

Trong Go, package encoding/json dùng để encode (generate) và decode (parse) JSON. Để chuyển struct sang JSON, dùng json.Marshal. Ngược lại, để parse JSON vào struct, dùng json.Unmarshal.

encoding/json giúp làm việc với JSON dễ dàng và trực tiếp trên struct mà không cần map thủ công.

"Dùng json.Marshal để chuyển struct sang JSON, json.Unmarshal để parse JSON vào struct. Dùng tag json:"fieldname" để map field."

21. How do you use the "reflect" package to inspect the type and value of a variable in Go?

Trong Go, package reflect cho phép bạn kiểm tra kiểu dữ liệu và giá trị của một biến trong runtime.

reflect.TypeOf trả về kiểu của biến, còn reflect.ValueOf trả về một object Value có thể dùng để lấy dữ liệu hoặc thao tác động trên biến. Package này hữu ích khi bạn viết code generic, kiểm tra struct field, hoặc thực hiện các thao tác runtime.

"Dùng reflect.TypeOf(var) để lấy kiểu và reflect.ValueOf(var) để lấy giá trị. Hữu ích cho inspect runtime và code generic."

22. How do you use the "testing" package to write unit tests in Go?

Trong Go, package testing dùng để viết unit test. Mỗi hàm test bắt đầu bằng Test và nhận tham số t *testing.T

Sau đó chạy test bằng lệnh go test. Bạn cũng có thể dùng t.Fatal, t.Error, hoặc tạo benchmark với testing.B. testing giúp đảm bảo code đúng chức năng và dễ dàng tự động kiểm tra khi refactor.

*"Viết hàm test bắt đầu bằng Test nhận t *testing.T. Dùng t.Errorf hoặc t.Fatal để kiểm tra kết quả. Chạy test bằng go test."*

23. How do you use the "errors" package to create and manipulate errors in Go?

Trong Go, package errors dùng để tạo và xử lý lỗi. Cách đơn giản nhất là dùng errors.New để tạo lỗi mới

Ngoài ra, từ Go 1.13, bạn có thể dùng fmt.Errorf với %w để wrap lỗi

Package errors cũng hỗ trợ errors.Is và errors.As để kiểm tra hoặc unwrap lỗi, giúp quản lý lỗi theo cách rõ ràng và an toàn.

"Dùng errors.New để tạo lỗi, fmt.Errorf("%w") để wrap lỗi. Dùng errors.Is hoặc errors.As để kiểm tra và unwrap lỗi."

24. How do you use the "net" package to implement networking protocols in Go?

Trong Go, package net cung cấp API để làm việc với TCP, UDP, IP và các protocol cơ bản. Package net cung cấp các interface như Conn, Listener, giúp đọc/ghi dữ liệu, quản lý kết nối, xây dựng server hoặc client cho nhiều protocol. Ngoài TCP/UDP, bạn cũng có thể dùng net để resolve DNS, IP, và làm việc với sockets.

"Dùng net.Listen để tạo server TCP/UDP, net.Dial để tạo client. net cung cấp Conn, Listener để đọc/ghi dữ liệu và quản lý kết nối."

25. How do you use the "time" package to handle dates and times in Go?

Trong Go, package time cung cấp các hàm và kiểu để làm việc với thời gian.
Ngoài ra, time còn hỗ trợ đo khoảng thời gian với time.Duration, tạo timer hoặc ticker, ví dụ time.Sleep(2 * time.Second) hoặc time.NewTicker(time.Second). Đây là cách Go quản lý date, time và scheduling rất tiện lợi.

"Dùng time.Now() để lấy thời gian hiện tại, Format/Parse để định dạng, time.Sleep hoặc Ticker để xử lý delay và lặp theo thời gian."

26. How do you use the "math" and "math/rand" packages to perform mathematical and statistical operations in Go?

Trong Go, package math cung cấp các hàm toán học chuẩn như Sqrt, Pow, Sin, Cos, v.v.
Package math/rand dùng để sinh số ngẫu nhiên
Kết hợp cả hai package giúp thực hiện tính toán, mô phỏng, và các thao tác thống kê cơ bản.

"Dùng math cho các hàm toán học như Sqrt, Pow, Sin, và math/rand để sinh số ngẫu nhiên với rand.Intn, rand.Float64, nhớ seed với rand.Seed."

27. How do you use the "crypto" package to perform cryptographic operations in Go?

Trong Go, package crypto và các subpackage của nó (crypto/sha256, crypto/hmac, crypto/rand, crypto/rsa, v.v.) cho phép thực hiện các thao tác mã hóa, băm, và xác thực.
Để sinh khóa RSA, ký dữ liệu hoặc tạo HMAC, Go cung cấp các subpackage chuyên dụng như crypto/rsa và crypto/hmac. Package crypto/rand cung cấp nguồn random an toàn để sinh khóa hoặc nonce. Kết hợp các gói này giúp thực hiện bảo mật và mã hóa hiệu quả.

"Dùng các subpackage của crypto như sha256, rsa, hmac, rand để băm, ký, sinh khóa, và tạo dữ liệu ngẫu nhiên an toàn."

28. How do you use the "os" package to interact with the operating system in Go?

Trong Go, package os cung cấp các hàm để tương tác với hệ điều hành, như làm việc với file, thư mục, environment variables, hoặc process.
os là package cơ bản để làm việc với hệ thống từ Go.

"Dùng os để mở, tạo, xóa file, đọc thư mục, truy xuất biến môi trường, và làm việc với process, ví dụ os.Open, os.Create, os.Getenv."

29. How do you use the "bufio" package to read and write buffered data in Go?

Trong Go, package bufio cung cấp buffered I/O để đọc và ghi dữ liệu hiệu quả, đặc biệt khi làm việc với file hoặc network.
bufio giúp giảm số lần truy cập hệ thống, cải thiện hiệu năng khi xử lý nhiều dữ liệu.
"Dùng bufio.Scanner để đọc dữ liệu theo dòng hoặc từ input khác, dùng bufio.Writer để ghi dữ liệu có buffer, nhớ gọi Flush()."

30. How do you use the "strings" package to manipulate strings in Go?

Trong Go, package strings cung cấp nhiều hàm tiện ích để xử lý chuỗi. strings giúp thao tác với chuỗi nhanh chóng, gọn gàng và hiệu quả.
"Dùng package strings để xử lý chuỗi: chuyển đổi case (ToUpper/ToLower), kiểm tra chứa (Contains), thay thế (ReplaceAll), tách (Split) và trim."

31. How do you use the "bytes" package to manipulate byte slices in Go?

Trong Go, package bytes cung cấp nhiều hàm tiện ích để xử lý byte slices.
bytes hữu ích khi bạn cần thao tác trực tiếp trên byte slices thay vì string, ví dụ khi xử lý dữ liệu nhị phân, network hoặc file.
"Dùng bytes để thao tác byte slices: kiểm tra chứa (Contains), chuyển case (ToUpper), thay thế (ReplaceAll), tách (Split). Thích hợp cho dữ liệu nhị phân hoặc network."

32. How do you use the "encoding/binary" package to encode and decode binary data in Go?

Trong Go, package encoding/binary dùng để encode và decode dữ liệu nhị phân theo endian cụ thể.
Package này hữu ích khi làm việc với file nhị phân, network protocol, hoặc bất cứ trường hợp nào cần serialize dữ liệu thành byte stream.
"Dùng binary.Write để encode giá trị vào buffer theo endian, và binary.Read để decode từ buffer. Thích hợp cho file nhị phân hoặc network data."

33. How do you use the "compress/gzip" package to compress and decompress data using the gzip algorithm in Go?

Trong Go, package compress/gzip cho phép nén và giải nén dữ liệu theo chuẩn gzip.
Package này hữu ích khi muốn giảm dung lượng dữ liệu lưu trữ hoặc truyền qua network.
"Dùng gzip.NewWriter để nén dữ liệu vào buffer, dùng gzip.NewReader để giải nén. Phù hợp cho nén lưu trữ hoặc truyền dữ liệu."

34. How do you use the "database/sql" package to access a SQL database in Go?

Trong Go, package database/sql cung cấp interface chuẩn để làm việc với các cơ sở dữ liệu SQL. Cần import driver tương ứng.
database/sql cung cấp các phương thức như Query, QueryRow, Exec, và hỗ trợ transaction.
Nó giúp code database chuẩn hóa và dễ thay đổi driver.
"Dùng sql.Open để kết nối, QueryRow hoặc Query để truy vấn, Exec để thực thi lệnh. Cần import driver database tương ứng và nhớ defer db.Close()."

35. How do you use the "html/template" package to generate HTML templates in Go?

Trong Go, package html/template dùng để tạo và render HTML một cách an toàn, tránh XSS. Template được tạo từ string hoặc file, sau đó gọi Execute hoặc ExecuteTemplate để truyền dữ liệu.

Package này rất hữu ích khi làm web server, kết hợp với net/http để render HTML động một cách an toàn.

"Dùng template.New và Parse để tạo template, rồi Execute hoặc ExecuteTemplate với dữ liệu để render HTML an toàn."

ADVANCED

1. Explain byte and rune types and how they are represented.

Trong Go, byte và rune là hai kiểu dữ liệu cơ bản để biểu diễn ký tự, nhưng chúng khác nhau về phạm vi và mục đích.

byte là alias của uint8, dùng để biểu diễn một giá trị 8-bit. Nó thường dùng để xử lý dữ liệu nhị phân hoặc ASCII.

rune là alias của int32, dùng để biểu diễn một Unicode code point. Vì Unicode có thể dùng nhiều byte hơn 8-bit, rune giúp biểu diễn mọi ký tự Unicode.

Khi bạn thao tác với string trong Go, string thực sự là slice của byte. Nếu muốn xử lý ký tự Unicode, bạn nên convert string thành slice of rune để tránh lỗi khi xử lý ký tự đa byte.

"byte là uint8 để lưu 1 byte, thường cho ASCII hoặc dữ liệu nhị phân. rune là int32 để lưu 1 Unicode code point, dùng cho ký tự đa byte."

2. You have developed a Go program on Linux and want to compile it for both Windows and Mac. Is it possible?

Go hỗ trợ cross-compilation rất tốt. Bạn có thể biên dịch chương trình trên Linux để chạy trên Windows hoặc macOS bằng cách set hai biến môi trường GOOS và GOARCH trước khi chạy go build.

Không cần cài đặt thêm toolchain cho hệ điều hành đích. Đây là một tính năng rất mạnh của Go, giúp phát triển đa nền tảng dễ dàng.

"Go hỗ trợ cross-compilation. Chỉ cần set GOOS và GOARCH trước go build để biên dịch cho Windows (GOOS=windows) hoặc macOS (GOOS=darwin)."

3. How can you compile a Go program for Windows and Mac?

Để biên dịch một chương trình Go cho Windows hoặc Mac, bạn có thể dùng cross-compilation bằng cách set các biến môi trường GOOS và GOARCH trước khi chạy lệnh go build.

GOOS xác định hệ điều hành đích (windows, darwin cho macOS, linux, v.v.), còn GOARCH xác định kiến trúc (amd64, arm64). Không cần cài thêm toolchain của hệ điều hành đích, Go sẽ tự động tạo binary phù hợp.

"Set GOOS và GOARCH trước go build. Ví dụ: GOOS=windows GOARCH=amd64 go build cho Windows, GOOS=darwin GOARCH=amd64 go build cho Mac."

4. Starting from an empty file, how would you create a Go program's basic structure? Annotate the most important parts of the source code using comments.

Trong Go, chương trình cơ bản gồm package chính, import các package cần thiết, và hàm main.

Các phần quan trọng:

- package main – bắt buộc để tạo executable, không dùng cho thư viện.
- import – khai báo các package cần sử dụng.
- func main() – entry point, Go sẽ chạy hàm này khi chương trình bắt đầu.
- Các comment giúp giải thích code và phần quan trọng cho người đọc.

"Tạo file Go với package main, import các package cần thiết, và định nghĩa func main() làm entry point. Ví dụ dùng fmt.Println để in ra màn hình."

5. What is the string data type in Golang, and how is it represented?

Trong Go, string là một kiểu dữ liệu cơ bản dùng để lưu trữ chuỗi ký tự. Mỗi string thực ra là một slice của byte bất biến, nghĩa là khi tạo một string mới, bạn không thể thay đổi các byte bên trong nó.

Chuỗi trong Go được biểu diễn theo UTF-8, nên có thể chứa ký tự ASCII hoặc Unicode.

Để thao tác theo ký tự Unicode, bạn có thể chuyển string thành slice of rune:[]rune(s).

"string trong Go là slice bất biến của byte, lưu trữ dữ liệu UTF-8. Dùng []rune(s) để thao tác theo ký tự Unicode."

6. How can you format the Go source code in an idiomatic way?

Trong Go, cách chuẩn và idiomatic để format code là dùng công cụ gofmt, hoặc lệnh go fmt. Nó tự động căn chỉnh indentation, spacing, và style theo quy ước Go, giúp code đồng nhất giữa các lập trình viên.

IDE như VS Code hay GoLand cũng hỗ trợ format tự động khi lưu file. Việc dùng gofmt giúp tránh tranh cãi về style và duy trì code dễ đọc, idiomatic theo Go.

"Dùng gofmt hoặc go fmt để tự động format code theo chuẩn Go, giúp code đồng nhất và idiomatic."

7. Can you change a specific character in a string?

Trong Go, string là immutable, nghĩa là bạn không thể trực tiếp thay đổi một ký tự trong string. Nếu muốn chỉnh sửa, bạn cần chuyển string thành slice of rune hoặc byte, thay đổi ký tự trong slice, rồi tạo lại string.

Cách này đảm bảo bạn có thể chỉnh sửa ký tự bất kỳ, bao gồm cả ký tự Unicode đa byte, mà vẫn giữ tính bất biến của string gốc.

"Không thể trực tiếp thay đổi ký tự trong string vì string immutable. Chuyển string thành slice of rune hoặc byte, chỉnh sửa slice, rồi chuyển lại thành string."

8. How can you concatenate string values? What happens when concatenating strings?

Trong Go, để nối (concatenate) string, bạn có thể dùng toán tử +.
Khi nối string, Go sẽ tạo một string mới vì string là immutable. Điều này nghĩa là mỗi lần nối, một vùng nhớ mới được cấp phát để chứa kết quả, không thay đổi string gốc.
Nếu cần nối nhiều string liên tục, đặc biệt trong vòng lặp, tốt hơn nên dùng strings.Builder để tránh cấp phát bộ nhớ nhiều lần.
"Dùng `+` để nối string. Vì string immutable, mỗi lần nối tạo một string mới. Dùng `strings.Builder` khi nối nhiều chuỗi liên tục để tối ưu hiệu năng."

9. Give an example of an array and slice declaration.

Trong Go, array có kích thước cố định, còn slice là mảng động, có thể thay đổi độ dài.
Array declaration – kích thước cố định, phải xác định khi khai báo
Slice declaration – có độ dài linh hoạt, có thể append phần tử
Sử dụng array khi biết trước số phần tử, dùng slice khi cần mảng linh hoạt.
"Array có kích thước cố định: var arr [5]int. Slice linh hoạt: s := []int{1,2} và có thể dùng append để thêm phần tử."

10. Explain the backing array of a slice value.

Trong Go, một slice là một view trên một mảng nền (backing array). Slice lưu giữ ba thông tin: pointer đến mảng nền, độ dài (len) và sức chứa (cap).
Khi bạn thay đổi giá trị trong slice, giá trị trong mảng nền cũng thay đổi
Sức chứa (cap) của slice là số phần tử còn lại từ vị trí bắt đầu của slice đến cuối mảng nền.
Điều này giúp slice tiết kiệm bộ nhớ, nhưng cũng có thể gây side-effect nếu nhiều slice cùng dùng chung mảng nền.
"Slice là view trên một mảng nền, lưu pointer, len và cap. Thay đổi slice sẽ ảnh hưởng mảng nền, và cap là số phần tử còn lại từ slice đến cuối array."

11. Explain the Golang map type and its advantages.

Trong Go, map là một kiểu dữ liệu built-in dùng để lưu trữ các cặp key-value. Key là duy nhất trong map và có thể là bất kỳ kiểu dữ liệu comparable nào, còn value có thể là bất kỳ kiểu nào.
Ưu điểm của map trong Go:
- Truy cập, thêm, xóa phần tử nhanh, gần như O(1).
- Key duy nhất, tránh trùng lặp.
- Dễ sử dụng với cú pháp gọn nhẹ (make, map[key] = value).
- Hữu ích khi cần tra cứu, thống kê, hoặc lưu trữ dữ liệu theo cặp key-value.
"Map là kiểu key-value, key duy nhất. Truy cập, thêm, xóa nhanh, cú pháp gọn nhẹ, thích hợp cho lookup hoặc lưu trữ dữ liệu có khóa."

12. What is the recommended Golang package for basic operations on files? What other Golang packages are used to work with files?

Trong Go, package cơ bản để làm việc với file là os. os cung cấp các hàm như os.Open, os.Create, os.Remove, os.ReadDir để đọc, ghi, tạo, xóa file và thư mục.
Ngoài os, còn một số package khác giúp thao tác file tiện lợi hơn:
io – hỗ trợ đọc và ghi dữ liệu từ file hoặc stream (io.Copy, io.ReadAll).
io/ioutil – trước Go 1.16, dùng để đọc/ghi file nhanh (ioutil.ReadFile, ioutil.WriteFile), hiện đã khuyến nghị dùng os và io thay thế.
bufio – đọc/ghi dữ liệu buffered, hữu ích khi xử lý file lớn hoặc theo dòng.
path/filepath – thao tác với đường dẫn file, join, clean, hoặc walk directory.
"Dùng os cho các thao tác cơ bản: mở, tạo, xóa file. Ngoài ra io, bufio, path/filepath hỗ trợ đọc/ghi, buffered I/O và xử lý đường dẫn."

13. Explain the object-oriented architecture of Golang.

Go không có class như các ngôn ngữ hướng đối tượng truyền thống, nhưng vẫn hỗ trợ lập trình hướng đối tượng thông qua structs và interfaces.
Structs: là kiểu dữ liệu do người dùng định nghĩa, dùng để nhóm các trường (fields) và có thể gắn các phương thức (methods) lên struct đó.
Interfaces: định nghĩa các method mà một kiểu dữ liệu phải implement. Go dùng cơ chế implicit interface, nghĩa là không cần khai báo rõ ràng implements.
Go cũng hỗ trợ composition thay vì inheritance, tức là struct có thể nhúng struct khác để tái sử dụng code.
Vậy nên, OOP trong Go dựa trên encapsulation qua struct, polymorphism qua interface, và composition thay vì kế thừa lớp truyền thống.
"Go uses structs for data + methods, interfaces for polymorphism, and composition instead of inheritance. It supports encapsulation and implicit interface implementation."

14. What is a struct type? Can you change the struct definition at runtime?

Trong Go, struct là kiểu dữ liệu do người dùng định nghĩa để nhóm nhiều trường (fields) lại với nhau, có thể chứa các kiểu dữ liệu khác nhau. Struct giúp tổ chức dữ liệu phức tạp và có thể gắn methods cho struct đó.
Thay đổi struct definition tại runtime? Không, struct là static type. Bạn không thể thêm hoặc xóa fields khi chương trình đang chạy. Nếu cần sự linh hoạt runtime, bạn có thể dùng map[string]interface{} hoặc reflect để thao tác dữ liệu động, nhưng struct gốc vẫn không thay đổi.
"Struct là kiểu dữ liệu nhóm các trường lại với nhau. Không thể thay đổi definition của struct tại runtime vì Go là static typed. Dùng map hoặc reflect nếu cần dữ liệu động."

15. What are anonymous structs and anonymous struct fields? Give an example of such a struct declaration.

Trong Go:

Anonymous struct là struct được định nghĩa trực tiếp mà không cần đặt tên type. Nó hữu ích khi bạn chỉ cần dùng struct một lần.

Anonymous struct field là khi một struct nhúng một struct khác mà không đặt tên trường, giúp tận dụng composition và các field của struct nhúng trở thành field của struct cha.

Anonymous struct field giúp giảm boilerplate, tái sử dụng struct khác mà không cần đặt tên field riêng.

"Anonymous struct là struct không tên, dùng trực tiếp; anonymous struct field là struct nhúng không tên, giúp composition và truy cập field trực tiếp."

16. Explain the defer statement in Golang. Give an example of a deferred function's call.

Trong Go, defer dùng để hoãn việc thực thi một hàm đến khi hàm hiện tại kết thúc. Nó rất hữu ích để giải phóng tài nguyên như đóng file, kết nối database hoặc unlock mutex, giúp code sạch và tránh rò rỉ tài nguyên.

Một điểm quan trọng: các lệnh defer được thực hiện theo LIFO (Last In First Out), nghĩa là lệnh defer cuối cùng được gọi trước."*

"Defer hoãn thực thi một hàm đến khi hàm hiện tại kết thúc. Thường dùng để đóng file, unlock mutex. Các defer được thực hiện theo LIFO."

17. What are the advantages of passing pointers to functions?

Khi truyền pointer vào một hàm trong Go, có một số lợi ích:

- Truyền tham chiếu, không copy toàn bộ dữ liệu: Nếu biến lớn (struct hoặc array lớn), truyền pointer tránh việc copy toàn bộ, tiết kiệm bộ nhớ và tăng hiệu năng.
- Hàm có thể thay đổi giá trị gốc: Khi truyền pointer, hàm có thể cập nhật giá trị của biến bên ngoài, điều này không thể làm nếu chỉ truyền giá trị.
- Tiết kiệm chi phí khi dùng struct hoặc array lớn: Vì chỉ truyền địa chỉ, tránh overhead khi copy dữ liệu lớn.

pointer giúp vừa tiết kiệm bộ nhớ, vừa cho phép hàm thao tác trực tiếp trên dữ liệu gốc.

"Truyền pointer giúp hàm thay đổi giá trị gốc, tránh copy dữ liệu lớn, tiết kiệm bộ nhớ và tăng hiệu năng."

18. What are Golang methods?

Trong Go, method là một hàm được gắn với một kiểu dữ liệu, thường là struct, nhờ vậy hàm đó có thể thao tác trực tiếp với dữ liệu của struct. Phương thức được định nghĩa bằng cách thêm receiver vào khai báo hàm.

Methods giúp triển khai tính năng OOP, kết hợp với interfaces để hỗ trợ polymorphism.

"Method là hàm gắn với một kiểu dữ liệu, thường là struct, cho phép thao tác trên dữ liệu của struct và hỗ trợ OOP với interface."

19. How can you ensure a Go channel never blocks while sending data to it?

Trong Go, gửi dữ liệu vào channel có thể block nếu channel unbuffered hoặc đầy. Để đảm bảo gửi không block, bạn có thể:

1. Sử dụng buffered channel: Tạo channel với dung lượng (capacity) đủ lớn
2. Sử dụng select với default: Gửi dữ liệu theo kiểu non-blocking, nếu không gửi được thì đi vào nhánh default. Cách này đảm bảo chương trình không bị treo khi channel đầy hoặc chưa có receiver.

"Để channel không block khi gửi dữ liệu, dùng buffered channel đủ dung lượng hoặc dùng select với default để gửi non-blocking."

20. Explain why concurrency is not parallelism?

Trong lập trình, concurrency và parallelism khác nhau:

Concurrency là khả năng chương trình quản lý nhiều công việc cùng lúc, nhưng không nhất thiết các công việc này chạy đồng thời. Go thể hiện concurrency qua goroutines, scheduler có thể hoán đổi các goroutines trên một hoặc nhiều CPU core.

Parallelism là khi nhiều công việc thực sự chạy đồng thời trên nhiều CPU core cùng một lúc.

Ví dụ: bạn có 10 goroutines chạy trên máy 1 core, chúng thực thi concurrently nhưng không thực sự parallel; nếu máy có 4 core, có thể một số goroutines chạy đồng thời, tạo parallelism.

Vậy nên, concurrency là về thiết kế để quản lý nhiều công việc, còn parallelism là thực hiện cùng lúc trên nhiều core.

"Concurrency = quản lý nhiều công việc cùng lúc; parallelism = thực hiện đồng thời trên nhiều core. Concurrency không đồng nghĩa với parallelism."

21. What is a data race?

Trong Go, data race xảy ra khi hai hoặc nhiều goroutines cùng truy cập một biến chia sẻ mà ít nhất một goroutine viết biến đó, và không có cơ chế đồng bộ bảo vệ, như mutex.

Data race là lỗi khó phát hiện vì kết quả phụ thuộc vào timing của goroutines. Go cung cấp công cụ -race để phát hiện data race khi chạy.

Để tránh data race, dùng sync.Mutex, channel, hoặc các cơ chế đồng bộ khác.

"Data race là khi nhiều goroutine truy cập cùng một biến, ít nhất một goroutine ghi biến mà không dùng cơ chế đồng bộ, dẫn đến lỗi không xác định."

22. How can you detect a data race in Go code?

Trong Go, bạn có thể phát hiện data race bằng cách sử dụng race detector tích hợp sẵn. Khi chạy chương trình, thêm flag -race vào lệnh go run hoặc go test.

Race detector sẽ báo vị trí xảy ra data race trong code, bao gồm goroutine nào truy cập biến và dòng code liên quan. Đây là cách chuẩn để debug các vấn đề concurrency và tránh lỗi không xác định.

"Dùng race detector của Go bằng cách chạy go run -race hoặc go test -race để phát hiện data race."

23. What is a Go channel? What operations are available on the channel type?

Trong Go, channel là kiểu dữ liệu dùng để truyền dữ liệu giữa các goroutine, hỗ trợ đồng bộ và truyền thông tin an toàn. Bạn có thể tưởng tượng channel giống như một đường ống mà một goroutine gửi dữ liệu và goroutine khác nhận dữ liệu.

Các operations cơ bản trên channel:

1. Send – gửi giá trị vào channel: ch <- value
2. Receive – nhận giá trị từ channel: val := <-ch
3. Close – đóng channel khi không còn gửi thêm dữ liệu: close(ch)
4. Range over channel – lặp qua các giá trị trong channel cho đến khi channel bị đóng
5. Select statement – chọn giữa nhiều channel để gửi hoặc nhận dữ liệu không block.

Channel giúp thiết kế các chương trình concurrent an toàn mà không cần mutex, đồng thời làm code gọn hơn.

"Channel là kênh truyền dữ liệu giữa goroutine. Các operations: gửi ch <- val, nhận val := <-ch, đóng close(ch), lặp for v := range ch, và select để xử lý nhiều channel."

24. How can you ensure that a goroutine in Go receives data from a channel without blocking indefinitely if there's no data available, and without terminating prematurely if the channel is still producing data?

Để goroutine nhận dữ liệu từ channel mà không bị block vô hạn khi chưa có dữ liệu, nhưng không kết thúc sớm nếu channel vẫn sản xuất dữ liệu, bạn có thể dùng select statement kết hợp default hoặc timeout, hoặc kết hợp range với channel đóng đúng cách.

- select cho phép non-blocking và kiểm soát linh hoạt.
- range đảm bảo goroutine tiếp tục nhận dữ liệu đến khi channel đóng, tránh kết thúc sớm.

"Dùng range trên channel để nhận dữ liệu đến khi channel đóng, hoặc dùng select với default để non-blocking, đảm bảo goroutine không block vô hạn và không kết thúc sớm."

25. Write a Golang program that declares a string variable, prints the address of the variable, declares another int variable, and a pointer to it.

```go
package main

import "fmt"

func main() {
    // Khai báo một biến string
    str := "Hello, Go!"
    fmt.Println("String value:", str)
    fmt.Println("Address of string variable:", &str)

    // Khai báo một biến int
    num := 42
    fmt.Println("\nInteger value:", num)

    // Khai báo một con trỏ trỏ tới biến num
    ptr := &num
    fmt.Println("Pointer to num:", ptr)
    fmt.Println("Value at pointer:", *ptr)
}
```

26. Create a Go program that defines a named type and a method (receiver function) for that type.

```go
package main

import "fmt"

// Định nghĩa một named type
type Circle struct {
    radius float64
}

// Định nghĩa method (receiver function) cho Circle
func (c Circle) Area() float64 {
    return 3.14 * c.radius * c.radius
}

func main() {
    c := Circle{radius: 5}
    fmt.Println("Radius:", c.radius)
    fmt.Println("Area:", c.Area())
}
```

27. Write a simple Golang program that uses a goroutine and a channel.

```go
package main

import (
    "fmt"
)

func sendData(ch chan string) {
    ch <- "Hello from goroutine" // gửi dữ liệu vào channel
}

func main() {
    ch := make(chan string) // tạo channel

    go sendData(ch) // chạy goroutine gửi dữ liệu

    msg := <-ch // nhận dữ liệu từ channel
    fmt.Println(msg) // in ra: Hello from goroutine
}
```

28. Write a program to find the sum of all even numbers from 1 to 100.

```go
package main

import "fmt"

func main() {
    sum := 0

    for i := 1; i <= 100; i++ {
        if i%2 == 0 { // kiểm tra số chẵn
            sum += i
        }
    }

    fmt.Println("Sum of even numbers from 1 to 100 is:", sum)
}
```

29. Write a program to find the largest and smallest elements in an array.

```go
package main

import "fmt"

func main() {
    arr := []int{12, 45, 2, 67, 34, 89, 1, 10}

    if len(arr) == 0 {
        fmt.Println("Array is empty")
        return
    }

    max := arr[0]
    min := arr[0]

    for _, v := range arr {
        if v > max {
            max = v
        }
        if v < min {
            min = v
        }
    }

    fmt.Println("Largest element:", max)
    fmt.Println("Smallest element:", min)
}
```

30. Write a program to check if a number is prime.

```go
package main

import (
    "fmt"
    "math"
)

func isPrime(n int) bool {
    if n <= 1 {
        return false
    }
    if n == 2 {
        return true
    }
    for i := 2; i <= int(math.Sqrt(float64(n))); i++ {
        if n%i == 0 {
            return false
        }
    }
    return true
}

func main() {
    num := 29

    if isPrime(num) {
        fmt.Println(num, "is a prime number")
    } else {
        fmt.Println(num, "is not a prime number")
    }
}
```

31. Write a program to calculate the factorial of a number using recursion.

```go
package main

import "fmt"

// Hàm tính giai thừa bằng đệ quy
func factorial(n int) int {
    if n == 0 || n == 1 {
        return 1
    }
    return n * factorial(n-1)
}

func main() {
    num := 5
    fmt.Printf("Factorial of %d is %d\n", num, factorial(num))
}
```

32. Write a program to count the number of words in a given string.

```go
package main

import (
    "fmt"
    "strings"
)

func main() {
    str := "Go is a simple and powerful language"

    // Tách chuỗi thành các từ
    words := strings.Fields(str)

    fmt.Println("String:", str)
    fmt.Println("Number of words:", len(words))
}
```

33. Write a program to generate random numbers within a specified range.

```go
package main

import (
    "fmt"
    "math/rand"
    "time"
)

func main() {
    // Khởi tạo seed để sinh số ngẫu nhiên khác nhau mỗi lần chạy
    rand.Seed(time.Now().UnixNano())

    min := 10
    max := 50

    // Sinh số ngẫu nhiên trong khoảng [min, max]
    randomNum := rand.Intn(max-min+1) + min

    fmt.Printf("Random number between %d and %d: %d\n", min, max, randomNum)
}
```

34. Write a program to implement a bubble sort algorithm.

```go
package main

import "fmt"

func bubbleSort(arr []int) {
    n := len(arr)
    for i := 0; i < n-1; i++ {
        for j := 0; j < n-i-1; j++ {
            if arr[j] > arr[j+1] {
                // Hoán đổi hai phần tử
                arr[j], arr[j+1] = arr[j+1], arr[j]
            }
        }
    }
}

func main() {
    arr := []int{64, 34, 25, 12, 22, 11, 90}
    fmt.Println("Original array:", arr)

    bubbleSort(arr)
    fmt.Println("Sorted array:", arr)
}
```

35. Write a program to generate permutations of a given string.

```go
package main

import "fmt"

// Hàm đệ quy sinh permutations
func permute(s string, l int, r int) {
    if l == r {
        fmt.Println(s)
    } else {
        for i := l; i <= r; i++ {
            s = swap(s, l, i)
            permute(s, l+1, r)
            s = swap(s, l, i) // backtrack
        }
    }
}

// Hàm hoán đổi 2 ký tự trong string
func swap(s string, i, j int) string {
    runes := []rune(s)
    runes[i], runes[j] = runes[j], runes[i]
    return string(runes)
}

func main() {
    str := "ABC"
    fmt.Println("Permutations of", str, ":")
    permute(str, 0, len(str)-1)
}
```

36. Write a program to implement the Boyer-Moore string search algorithm.

37. Write a program to find the longest increasing subsequence in a given array.

38. Write a program to implement a binary tree and perform various operations.

39. Write a program to implement a stack using an array.

40. Write a program to implement a queue using a linked list.