DOMINANDO GO EM 1 ANO

Caderno de Exercícios

52 aulas de prática, do #01 ao #52

Go 1.27 · 9 módulos · 122 questões · 6 desafios integradores

Autor Desconhecido · theknowledgeisle.com

Nome do aluno: Início: Término:

Como usar este caderno

Cada módulo reúne um bloco de aulas do curso, com a lista dos artigos que ele cobre. As perguntas conceituais pedem explicações com as suas palavras; os exercícios de código pedem programas: escreva, rode com `go run` e teste com `go test -race`.

Tente sempre resolver sozinho antes de abrir o **Caderno de Respostas**. É no erro que o conhecimento fixa.

Sumário

Módulo	Tema	Aulas	Questões
MÓDULO 1	Fundamentos da Linguagem	#01–#08	Q1–Q17
MÓDULO 2	Estruturas de Dados e Coleções	#09–#14	Q18–Q30
MÓDULO 3	Pacotes, Módulos e Organização	#15–#18	Q31–Q42
MÓDULO 4	Concorrência	#19–#25	Q43–Q57
MÓDULO 5	I/O, HTTP e APIs	#26–#31	Q58–Q69
MÓDULO 6	Banco de Dados	#32–#37	Q70–Q82
MÓDULO 7	Ferramentas, CLI e Qualidade	#38–#43	Q83–Q94
MÓDULO 8	Arquitetura e Padrões	#44–#48	Q95–Q108
MÓDULO 9	Deploy, Cloud e Carreira	#49–#52	Q109–Q122
DESAFIOS	Desafios integradores	todas	D1–D6

Legenda: nas perguntas, `assim` indica código, nomes de tipos e comandos. As linhas pautadas são para a sua resposta; nos exercícios de código, use também o computador: **todo programa deve ser compilado e executado**.

MÓDULO 1

Fundamentos da Linguagem

Artigos 01 a 08

#01 Introdução ao Go: história, filosofia e os criadores da linguagem
#02 Instalação, configuração do ambiente e o comando `go`
#03 Variáveis, tipos primitivos e declaração curta
#04 Operadores, expressões e conversão de tipo
#05 Estruturas de controle: if, for e switch
#06 Funções: declaração, múltiplos retornos e variádicas
#07 Ponteiros: conceito, uso e quando evitar
#08 Arrays e Slices: a espinha dorsal das coleções em Go

Perguntas conceituais

Q1 Quem são os três criadores do Go e em que empresa trabalhavam quando projetaram a linguagem?

Q2 O que é a **promessa de compatibilidade do Go 1** (2012)? O que ela garante, o que fica de fora dela e qual é o papel da linha `go` do `go.mod` quando a linguagem muda o comportamento de algo, como aconteceu com a variável de laço no Go 1.22?

Q3 Qual é a diferença entre `var x int` e `x := 0` no Go? Em quais escopos cada forma é permitida?

Q4 Qual é o valor zero de cada tipo: `int`, `float64`, `string`, `bool`, `*int`, `[]int`, `map[string]int`?

Q5 O que acontece ao tentar dividir dois inteiros em Go: `7 / 2`? Como obter `3.5`?

Q6 Por que `if x = 5 { }` não compila em Go, ao contrário de C? E por que o compilador também recusa uma variável local declarada e nunca usada, e um `import` que nada usa?

Q7 Por que `fmt.Printf("%d", "texto")` é detectado pelo `go vet` mas não impede a compilação?

Q8 Go não tem `while` nem `do-while`. Usando só `for`, escreva: (a) a soma de 1 a 100 no estilo `while`; (b) um laço que lê números de uma lista até encontrar `0` e soma os anteriores, no estilo `do-while` (o corpo roda pelo menos uma vez); (c) a soma de (a) com `range` sobre um inteiro.

Q9 Qual é a diferença entre `break` e `continue` dentro de um laço? Dê um exemplo de cada.

Q10 O que é `defer`? Em que ordem múltiplos `defer`s são executados? Dê um exemplo prático.

Exercícios de código

Q11 Complete o código abaixo para que a função retorne o maior dos três valores:

```
func maior(a, b, c int) int {
    // escreva aqui



}
```

Q12 Corrija o código abaixo (ele tem 4 erros):

```
package main

func main() {
    var x int = 10.5
    y := x + "5"
    fmt.Println(y)
    x++
    fmt.Println(x
}
```

Q13 Escreva uma função variádica `soma(nums ...int) int` que retorna a soma de todos os argumentos.

Q14 O que imprime o código abaixo? Explique por quê.

```
x := 42
p := &x
*p = 100
fmt.Println(x)
```

Q15 Qual a diferença entre um array `[5]int` e um slice `[]int`? Por que slices são mais usados no dia a dia?

Q16 O que acontece quando `append` precisa aumentar a capacidade de um slice? Explique o mecanismo de realocação.

Q17 Escreva uma função que receba um slice de strings e retorne um novo slice sem duplicatas, preservando a ordem.

MÓDULO 2

Estruturas de Dados e Coleções

Artigos 09 a 14

#09 Maps: criação, iteração e boas práticas
#10 Structs: definição, inicialização e campos embutidos
#11 Métodos em Structs: receivers por valor e por ponteiro
#12 Interfaces: contratos implícitos e polimorfismo
#13 Composição vs herança: o jeito Go de reutilizar código
#14 O tipo `error` e tratamento idiomático de falhas

Perguntas conceituais

Q18 Por que a ordem de iteração sobre um map em Go é não garantida? O que fazer quando a ordem importa?

Q19 Qual é a diferença entre declarar `var m map[string]int` e `m := make(map[string]int)`? O que acontece ao escrever em cada um?

Q20 Explique o idioma 'comma ok' para maps: `v, ok := m["chave"]`. Quando ele é necessário?

Q21 O que são campos embutidos (embedding) em structs? Como eles diferem de herança?

Q22 Qual é a diferença entre um receiver por valor `(c Conta)` e por ponteiro `(c *Conta)`? Quando usar cada um?

Q23 Por que Go usa implementação implícita de interfaces? Que vantagem isso traz em relação ao `implements` explícito de Java?

Q24 O que é o tipo `error`? Como criar erros customizados que carregam informações estruturadas?

Q25 Qual a diferença entre `errors.Is` e `errors.As`? Dê um exemplo de uso de cada um.

Q26 O que é wrapping de erro com `%w` no `fmt.Errorf`? Por que isso é importante para a rastreabilidade?

Q27 Explique `panic` e `recover`. Em que situações seu uso é apropriado versus o retorno normal de `error`?

Exercícios de código

Q28 Implemente um tipo `Pilha` genérica (stack) usando slice, com os métodos `Push`, `Pop` e `Tamanho`.

Q29 Crie uma interface `Forma` com os métodos `Area() float64` e `Perimetro() float64`. Implemente-a para `Retangulo` e `Circulo`. Escreva uma função `descrever(f Forma)` que imprime área e perímetro.

Q30 O que imprime o código abaixo? Existe algum problema? Corrija se necessário.

```
var i interface{} = (*int)(nil)
fmt.Println(i == nil)
```

MÓDULO 3

Pacotes, Módulos e Organização

Artigos 15 a 18

#15 Pacotes: visibilidade, nomenclatura e organização de projeto
#16 Go Modules: go.mod, go.sum e gerenciamento de dependências
#17 A biblioteca padrão: um tour pelas principais ferramentas
#18 Testes em Go: testing, tabelas de casos e benchmarks

Perguntas conceituais

Q31 Como Go controla visibilidade de identificadores? Qual a regra que determina se algo é exportado?

Q32 Para que serve o diretório `internal/`? Quem pode importar pacotes dentro dele?

Q33 Qual a diferença entre `go.mod` e `go.sum`? Por que ambos devem ser commitados no repositório?

Q34 O que faz `go mod tidy`? Quando deve ser executado?

Q35 Explique o Minimum Version Selection (MVS) do Go. Por que ele é preferível a 'pegar a versão mais recente'?

Q36 Cite 5 pacotes da biblioteca padrão e suas responsabilidades principais.

Q37 Qual a diferença entre `json.Marshal` e `json.NewEncoder`? Quando usar cada um?

Q38 O que é `time.After` e como ele difere de `time.NewTimer`? Qual o problema de usar `time.After` dentro de loops?

Q39 Como funciona o padrão de tabela de casos em testes Go (`table-driven tests`)? Quais são suas vantagens?

Q40 O que faz `t.Helper()` em uma função de teste auxiliar? Por que é importante?

Exercícios de código

Q41 Escreva um teste de tabela de casos para a função `Palindromo(s string) bool` cobrindo pelo menos 6 casos, incluindo strings com espaços, maiúsculas e caracteres especiais.

Q42 Escreva uma função de benchmark para medir a performance de duas implementações de contagem de palavras: uma usando `strings.Split` e outra usando `strings.Fields`.

MÓDULO 4

Concorrência

Artigos 19 a 25

#19 Goroutines: concorrência leve e o modelo de execução do Go
#20 Channels: comunicação entre goroutines
#21 Select, timeouts e cancelamento de operações
#22 sync.WaitGroup e sync.Mutex: coordenando goroutines
#23 O pacote context: propagação de cancelamento e prazos
#24 Padrões de concorrência: fan-out, fan-in e pipelines
#25 Armadilhas da concorrência: race conditions e como detectá-las

Perguntas conceituais

Q43 O que é uma goroutine? Em que ela difere de uma thread do sistema operacional em termos de uso de memória?

Q44 O que é o scheduler M:N do Go? Como ele mapeia goroutines para threads do SO?

Q45 Qual é a diferença entre um channel com buffer e sem buffer? Quando cada um deve ser usado?

Q46 O que é o padrão 'done channel'? Como `close(done)` difere de enviar um valor para `done`?

Q47 Explique o padrão de pipeline com channels. Quais são os papéis de produtor, estágio e consumidor?

Q48 O que é fan-out? O que é fan-in? Como esses padrões são implementados com goroutines e channels?

Q49 Qual a diferença entre `sync.Mutex` e `sync.RWMutex`? Em que cenário `RWMutex` oferece vantagem?

Q50 O que é uma race condition? Como o detector de races do Go (`-race`) funciona?

Q51 Por que `sync.Once` é thread-safe? Como ele garante que a função seja executada exatamente uma vez?

Q52 Qual é o propósito do pacote `context`? Explique a árvore de contextos e como o cancelamento se propaga.

Q53 O que é uma goroutine leak? Cite 3 causas comuns e como evitá-las.

Q54 O que é o problema da variável de loop capturada em closures? Como ele foi corrigido no Go 1.22?

Exercícios de código

Q55 Implemente uma função `buscarConcorrente(urls []string) []Resultado` que faz requisições HTTP para todas as URLs simultaneamente e coleta os resultados. Use WaitGroup e channel.

Q56 O código abaixo tem uma race condition. Identifique-a e corrija usando sync/atomic ou Mutex.

```
var contador int

for i := 0; i < 1000; i++ {
    go func() { contador++ }()
}
time.Sleep(time.Second)
fmt.Println(contador)
```

Q57 Implemente um semáforo usando channel com buffer que limita o número de goroutines simultâneas a N.

MÓDULO 5

I/O, HTTP e APIs

Artigos 26 a 31

#26 Leitura e escrita de arquivos com os pacotes os e bufio
#27 Trabalhando com JSON: encode, decode e tags de struct
#28 Criando um servidor HTTP com net/http
#29 Roteamento avançado com Gorilla Mux ou Chi
#30 Consumindo APIs externas com http.Client
#31 Middlewares: logging, autenticação e CORS

Perguntas conceituais

Q58 Qual a diferença entre `os.ReadFile` e `bufio.Scanner` para ler um arquivo? Quando usar cada abordagem?

Q59 Por que `bufio.Writer` melhora a performance de escrita? O que acontece se `Flush()` não for chamado?

Q60 Explique as tags de struct JSON: `json:"-"`, `json:"nome,omitempty"` e `json:",string"`. O que cada uma faz?

Q61 Qual a diferença entre `json.Marshal` e `json.MarshalIndent`? Quando `json.RawMessage` é útil?

Q62 Por que nunca usar `http.DefaultClient` em produção? O que pode acontecer sem timeout configurado?

Q63 Como o Go 1.22 melhorou o roteador `net/http`? Dê exemplos de padrões de rota que agora são suportados.

Q64 O que é um middleware em Go? Qual é sua assinatura padrão e como middlewares são encadeados?

Q65 Para que serve o pacote `net/http/httptest`? Como ele facilita testes de handlers HTTP?

Q66 O que é graceful shutdown de um servidor HTTP? Por que é importante e como é implementado?

Q67 Explique o padrão de retry com backoff exponencial e jitter. Por que o jitter é necessário?

Exercícios de código

Q68 Escreva um middleware de rate limiting simples que limita cada IP a 10 requisições por minuto, retornando HTTP 429 quando o limite é excedido.

Q69 Implemente uma função `lerCSVParaMap(caminho string) ([]map[string]string, error)` que lê um arquivo CSV e retorna cada linha como um mapa de nome-coluna → valor.

MÓDULO 6

Banco de Dados

Artigos 32 a 37

#32 Introdução ao database/sql e drivers em Go
#33 Go com MySQL e MariaDB: conexão, queries e transações
#34 Go com SQLite: banco de dados embutido para aplicações leves
#35 Go com PostgreSQL: recursos avançados e o driver pgx
#36 Go com MongoDB: operações CRUD com o driver oficial
#37 GORM: ORM para Go, migrações e relacionamentos

Perguntas conceituais

Q70 Por que nunca interpolar strings do usuário diretamente em queries SQL? Como placeholders (`?` ou `$1`) previnem SQL injection?

Q71 O que é o pool de conexões do `sql.DB`? Quais são as configurações mais importantes e o que cada uma controla?

Q72 Por que `rows.Close()` é obrigatório e deve usar `defer`? O que acontece se não for chamado?

Q73 Qual a diferença entre `QueryRowContext` e `QueryContext`? Quando usar cada um?

Q74 Explique o `_journal_mode=WAL` do SQLite. Que vantagem ele oferece sobre o modo padrão?

Q75 Por que `SetMaxOpenConns(1)` é recomendado para SQLite? O que acontece com escritas concorrentes sem isso?

Q76 Qual a diferença entre `lib/pq` e `pgx` como drivers PostgreSQL? Por que `pgx` é preferido para novos projetos?

Q77 Como transações ACID garantem consistência em operações que envolvem múltiplas tabelas? Dê um exemplo de transferência bancária.

Q78 O que é `bson.M` no MongoDB? Qual a diferença entre `bson.M` e `bson.D`?

Q79 O que é `AutoMigrate` no GORM? O que ele faz e o que ele NÃO faz (para não quebrar produção)?

Q80 O que é soft delete no GORM? Como o `gorm.Model` implementa isso? Como buscar registros deletados?

Exercícios de código

Q81 Escreva uma função `transferir(ctx, db, origemID, destinoID int, centavos int64) error` usando transação SQL que garante atomicidade, verifica saldo e faz rollback em caso de falha.

Q82 Escreva um repositório em memória que implementa uma interface `UsuarioRepository` com os métodos `Criar`, `BuscarPorID`, `Listar` e `Deletar`. Esse repositório deve ser thread-safe.

MÓDULO 7

Ferramentas, CLI e Qualidade

Artigos 38 a 43

#38 Criando ferramentas de linha de comando com flag e Cobra
#39 Logging estruturado com slog e Zerolog
#40 Variáveis de ambiente e configuração com Viper
#41 Linting, formatação e análise estática: gofmt, golangci-lint
#42 Documentação com godoc e boas práticas de comentário
#43 Profiling e otimização de performance em Go

Perguntas conceituais

Q83 Qual a diferença entre o pacote `flag` da biblioteca padrão e o Cobra? Em que situações cada um é adequado?

Q84 O que é a flag `--config` no Viper e como ela interage com variáveis de ambiente? Qual é a ordem de precedência?

Q85 Qual a diferença entre `log/slog` e o pacote `log` padrão? Por que logging estruturado é preferível em produção?

Q86 O que faz `slog.With`? Como ele é usado para adicionar contexto a todos os logs de uma requisição?

Q87 Por que o Zerolog usa encadeamento de métodos (`logger.Info().Str(...).Msg(...)`)? Que vantagem de performance isso oferece?

Q88 Quais são os linters mais importantes do `golangci-lint`? O que cada um detecta?

Q89 O que é `go vet`? Cite 3 problemas que ele detecta que o compilador não detecta.

Q90 Quais são as regras do `godoc` para comentários de documentação? Qual é a convenção para funções, tipos e erros?

Q91 O que é `go tool pprof`? Quais tipos de perfis ele suporta e como cada um é coletado?

Q92 O que é escape analysis? Como a flag `-gcflags="-m"` ajuda a identificar alocações no heap?

Exercícios de código

Q93 Usando Cobra, crie a estrutura de um comando `app usuario criar --nome "João" --email joao@x.com --papel admin` com validação dos campos obrigatórios.

Q94 Crie um benchmark que compara duas versões de uma função que conta ocorrências de uma substring: uma usando `strings.Count` e outra usando um loop manual. Use `b.ResetTimer()` corretamente.

MÓDULO 8

Arquitetura e Padrões

Artigos 44 a 48

#44 Clean Architecture em Go: camadas e dependências
#45 Domain-Driven Design aplicado a projetos Go
#46 Injeção de dependência sem frameworks
#47 Design Patterns em Go: Repository, Factory e Strategy
#48 APIs RESTful completas: projeto do zero ao deploy

Perguntas conceituais

Q95 Qual é o princípio central da Clean Architecture? Como as dependências devem apontar entre as camadas?

Q96 Qual a diferença entre uma Entidade e um Value Object no DDD? Dê exemplos de cada um em um sistema de e-commerce.

Q97 O que é um Aggregate Root no DDD? Por que todas as modificações do agregado devem passar por ele?

Q98 O que são Bounded Contexts no DDD? Por que o mesmo conceito (ex: 'Cliente') pode ter modelos diferentes em contextos diferentes?

Q99 O que são eventos de domínio? Como eles diferem de eventos de infraestrutura?

Q100 O que é injeção de dependência? Por que Go prefere a abordagem manual em vez de frameworks de DI?

Q101 O que é o padrão de opções funcionais? Que problema ele resolve em relação a construtores com muitos parâmetros?

Q102 O que é a Composition Root? Por que toda instanciação de dependências deve acontecer lá?

Q103 Qual é a diferença entre o padrão Repository e um DAO (Data Access Object)?

Q104 Explique o padrão Decorator em Go. Como ele é usado para adicionar cache e logging a um repositório sem modificar sua implementação?

Q105 O que é o padrão Strategy? Como Go o implementa de forma mais simples que linguagens com herança?

Q106 Como o padrão Observer é implementado de forma idiomática em Go? Qual a vantagem de usar goroutines para notificação assíncrona?

Exercícios de código

Q107 Projete as camadas de uma aplicação de blog seguindo Clean Architecture. Desenhe (ou descreva em texto) quais pacotes existem, suas responsabilidades e como as dependências fluem entre eles.

Q108 Implemente um Value Object `Email` em Go que valide o formato, seja imutável e implemente igualdade por valor.

MÓDULO 9

Deploy, Cloud e Carreira

Artigos 49 a 52

#49 Compilação cruzada, binários estáticos e Docker com Go
#50 Go na nuvem: deploy no Railway, Fly.io e Google Cloud Run
#51 gRPC e Protocol Buffers: comunicação eficiente entre serviços
#52 O ecossistema Go: comunidade, carreira e próximos passos

Perguntas conceituais

Q109 O que fazem as flags `-w` e `-s` no `go build`? Por que `-trimpath` é importante para builds reproduzíveis?

Q110 Como funciona a compilação cruzada no Go? Quais variáveis de ambiente controlam o sistema e arquitetura alvo?

Q111 O que é `CGO_ENABLED=0`? Em que situações é necessário? Qual o impacto em drivers de banco de dados?

Q112 O que é um build multistage no Docker? Qual a vantagem de usar `distroless` como imagem final?

Q113 Como o pacote `embed` funciona? O que as diretivas `//go:embed` permitem incluir no binário?

Q114 Quais são as diferenças fundamentais entre Railway, Fly.io e Google Cloud Run? Quando cada um é mais adequado?

Q115 O que são Protocol Buffers? Por que gRPC usa serialização binária em vez de JSON?

Q116 Quais são os 4 tipos de RPC no gRPC? Dê um exemplo de caso de uso para cada um.

Q117 O que são interceptors no gRPC? Como eles se comparam a middlewares HTTP?

Q118 O que é `grpc-gateway`? Que problema ele resolve para equipes que precisam suportar tanto REST quanto gRPC?

Q119 Cite 5 ferramentas ou projetos famosos escritos em Go e explique por que Go foi escolhido para cada um.

Q120 O que são os **iteradores** (*range-over-func*, Go 1.23)? Escreva `Pares(n int) iter.Seq[int]`, que produz os pares menores que `n`, use-a num `for range` e explique o que acontece com o iterador quando o corpo do laço dá `break`.

Exercícios de código

Q121 Escreva um Dockerfile multistage para uma API Go que: (1) compila com CGO_ENABLED=0, (2) usa distroless como imagem final, (3) embute versão e commit via ldflags, (4) roda como usuário não-root.

Q122 Defina em Proto3 um serviço `ProdutoService` com os métodos: `CriarProduto`, `BuscarProduto` (por ID), `ListarProdutos` (com paginação) e `AssistirEstoque` (streaming server-side que envia alertas quando estoque cai abaixo de um limite).

PARA IR ALÉM

Desafios integradores

Projetos maiores, que juntam vários módulos. Não há uma única resposta certa.

D1

API REST de Tarefas Completa

Construa uma API REST completa para gerenciamento de tarefas com: (a) Autenticação JWT com middleware; (b) CRUD de tarefas com PostgreSQL e pgx; (c) Paginação e filtros por status e prioridade; (d) Logging estruturado com slog; (e) Testes de integração usando banco SQLite em memória; (f) Dockerfile multistage. Descreva a estrutura de pacotes e as principais interfaces.

Sua solução (arquitetura, decisões e código principal):

D2

Worker Pool Configurável

Implemente um worker pool em Go que: (a) Receba tarefas via channel; (b) Processe com N workers configurável; (c) Respeite cancelamento via context; (d) Colete resultados e erros separadamente; (e) Exponha métricas (tarefas processadas, em fila, com erro); (f) Encerre graciosamente esperando workers terminarem. Escreva os testes de concorrência com -race.

Sua solução (arquitetura, decisões e código principal):

D3

Sistema de Cache com TTL

Projete e implemente um cache em memória thread-safe com: (a) Operações Get, Set, Delete; (b) TTL configurável por chave; (c) Eviction automática de entradas expiradas; (d) Limite máximo de entradas com política LRU; (e) Interface que permita substituição por Redis ou Memcached; (f) Benchmarks comparando com sync.Map.

Sua solução (arquitetura, decisões e código principal):

D4

CLI de Deploy

Crie uma ferramenta CLI usando Cobra que: (a) Tenha subcomandos `build`, `push` e `deploy`; (b) Leia configuração de arquivo YAML com Viper; (c) Execute compilação cruzada para múltiplos alvos; (d) Construa e empurre imagens Docker; (e) Faça deploy via API HTTP com retry e backoff; (f) Tenha output colorido e progress bars; (g) Gere completions para bash e zsh.

Sua solução (arquitetura, decisões e código principal):

D5

Microserviço gRPC com Streaming

Implemente um microserviço de monitoramento de preços que: (a) Exponha um serviço gRPC com streaming bidirecional; (b) Clientes subscrevem a alertas de produtos específicos; (c) Quando o preço muda, o servidor notifica todos os subscribers; (d) Use context para cancelamento e timeouts; (e) Implemente interceptors de autenticação e logging; (f) Exponha também uma interface REST via grpc-gateway.

Sua solução (arquitetura, decisões e código principal):

D6

Uma calculadora, duas interfaces

Escreva uma calculadora com sete operações: soma, subtração, multiplicação, divisão, potência (expoente inteiro), porcentagem (`15 % 200` é 15% de 200) e raiz quadrada. Ela deve: (a) fazer as contas com `math/big`, para que `0,1 + 0,2` dê exatamente `0,3`, aceitando vírgula ou ponto e mostrando até 16 dígitos significativos; (b) guardar o último resultado em `ans`, que pode ser usado na conta seguinte; (c) devolver um `error` claro para divisão por zero, raiz de negativo, expoente fracionário e texto que não é número, **sem encerrar o programa**. Escreva o núcleo num pacote que não sabe nada de entrada e saída, e reaproveite-o em dois programas: um de **terminal** (com os comandos `hist` e `sair`) e uma interface **gráfica no navegador**, servida com `net/http`, com teclado na tela, visor e histórico.

Sua solução (arquitetura, decisões e código principal):

Bom estudo e bons exercícios!

Dominando Go em 1 Ano · theknowledgeisle.com