Here’s a ready to copy Go “notes” string you can drop into a source file (e.g. as a `const` or a raw string literal).
Everything is kept in plain text, well sectioned, and free of any extra markup.

```go
// golangNotes holds a quick reference cheat sheet for the Go programming language.
const golangNotes = `
Go (Golang) Quick Reference
==========================================

1. Basics
---------
- Package declaration:   package main
- Entry point:           func main() { … }
- Imports:               import ("fmt" "net/http")
- Build & run:           go build ./...   |   go run main.go

2. Variables & Constants
------------------------
- Declaration:
    var i int           // zero value 0
    var s = "hello"
    var (
        a int = 1
```

```
        b = true
    )
```
- Short declaration (inside functions):
```
    i := 42
```
- Constants:
```
    const Pi = 3.14159
    const (
        StatusOK = 200
        StatusNotFound = 404
    )
```
- Typed constants (untyped constants can be used as any compatible type).

3. Types

- Primitive: bool, string, int, int8/16/32/64, uint, float32/64, byte (= uint8), rune (= int32)
- Composite:
```
    array:   var a [5]int
    slice:   s := []int{1,2,3}
    map:     m := map[string]int{"one":1}
    struct:  type Point struct{ X, Y float64 }
    channel: ch := make(chan int)          // unbuffered
    chBuf := make(chan int, 10)            // buffered
```
- Type alias: type MyInt = int
- New type: type MyInt int // distinct type

4. Functions

- Declaration:
  func Add(a, b int) int { return a + b }
- Multiple return values:
  func DivMod(a, b int) (quot, rem int) { return a / b, a % b }
- Variadic:
  func Sum(nums ...int) int { … }
- Named return values:
  func Split(sum int) (x, y int) { x = sum/2; y = sum - x; return }
- Defer (LIFO):
  defer file.Close()
- Panic / Recover:
  panic("boom")
  defer func(){ if r:=recover(); r!=nil { fmt.Println(r) } }()

5. Methods & Interfaces

- Method on a named type:
  type Counter int
  func (c *Counter) Inc() { *c++ }
- Interface:
  type Reader interface {
      Read(p []byte) (n int, err error)
  }
- Implicit implementation (no explicit “implements” keyword).

- Empty interface: interface{} (holds any value).

6. Structs & Embedding

```
    type Person struct {
        Name string
        Age  int
    }
    type Employee struct {
        Person      // anonymous field !’ promoted fields
        Salary float64
    }
```

- Constructor idiom:

```
    func NewEmployee(name string, age int, salary float64) *Employee {
        return &Employee{Person: Person{Name: name, Age: age}, Salary: salary}
    }
```

7. Concurrency

- Goroutine: go f()
- Channels:

```
    ch := make(chan int)          // unbuffered
    ch <- 42                    // send (blocks until receiver)
    v := <-ch                   // receive (blocks until sender)
```

- Select:
```
    select {
    case v := <-ch1:
        fmt.Println("ch1:", v)
    case ch2 <- 7:
        fmt.Println("sent to ch2")
    default:
        fmt.Println("no communication")
    }
```
- Context (cancellation, timeouts):
```
    ctx, cancel :=
context.WithTimeout(context.Background(),
2*time.Second)
    defer cancel()
    // pass ctx down the call chain
```

8. Packages & Modules

- Create a module:
```
    go mod init github.com/user/project
```
- Add a dependency:
```
    go get github.com/sirupsen/logrus@v1.9.0
```
- Internal packages: use `internal/` to restrict visibility.
- Exported identifiers start with an upper case letter.

9. Error Handling

- Idiomatic pattern:
    if err != nil {
        return fmt.Errorf("operation failed: %w", err) // wrap with %w (Go /1.13+)
    }
- Define sentinel errors:
    var ErrNotFound = errors.New("not found")
- Custom error type:
    type MyError struct{ Msg string }
    func (e *MyError) Error() string { return e.Msg }

10. Testing

- Test file naming: *_test.go
- Test function signature:
    func TestAdd(t *testing.T) { … }
- Table driven tests:
    tests := []struct{ a, b, want int }{{1,2,3},{-1,1,0}}
    for _, tt := range tests { … }
- Benchmarks:
    func BenchmarkAdd(b *testing.B) { for i:=0; i<b.N; i++ { Add(1,2) } }
- Coverage:
    go test -cover ./...

11. Common Tools

- Formatting: gofmt / go fmt

- Linting: golangci-lint
- Staticcheck: staticcheck
- Dependency vetting: go mod tidy, go mod verify
- Profiling: go tool pprof
- Race detector: go run -race main.go

12. Best Practices (quick list)

- Keep packages small & focused.
- Use interfaces to depend on abstractions, not implementations.
- Prefer early returns to deep nesting.
- Do not ignore errors (unless truly intentional).
- Use context for request scoped work.
- Write table driven tests.
- Document exported symbols (godoc comments).
- Use `go.mod` for versioned dependencies; avoid vendoring unless needed.
- Keep the public API stable; deprecate with a comment before removal.

==========
End of Go Quick Reference
==========
`
```
```