Go Cheatsheet

Basic Structure and Execution

A Go program consists of a package declaration, imports, and functions.

```
package main // Package declaration

import "fmt" // Import standard library

// Program entry point
func main() {
    fmt.Println("Hello, World!")
}
```

- Go CLI:
  - `go run main.go`: Run the program immediately.
  - `go build main.go`: Compile and generate an executable.
  - `go install`: Compile and install the package to the bin directory.
  - `go mod init <module_name>`: Initialize a new Go module.
  - `go mod tidy`: Add missing modules and remove unused ones.

Variables and Data Types

- Declaration and Initialization:
  - `var x int = 10`: Explicit type declaration.
  - `y := 20`: Short variable declaration (type inference, only inside functions).
  - `const PI = 3.14`: Declare a constant.
- Basic Types: `int`, `float64`, `bool`, `string`, `complex128`.
- Zero Values: `0` for numbers, `false` for booleans, `""` for strings, `nil` for pointers/collections.

Collections

Array and Slice

- Array (fixed size): `var arr [5]int = [5]int{1, 2, 3, 4, 5}`
- Slice (dynamic size):
  - `s := []int{1, 2, 3}`
  - `s = append(s, 4)`: Add an element.
  - `s2 := make([]int, 5, 10)`: Create with length 5 and capacity 10.

Map

- `m := make(map[string]int)`
- `m["key"] = 100`
- `val, ok := m["key"]`: Access a value and check for existence.

Control Flow

- `if-else`:

```
if x > 10 {
    // Logic
} else {
    // Logic
}

// Initialization inside if statement
if val := compute(); val < 0 { ... }
```

- `for` Loop (The only loop statement in Go):
  - `for i := 0; i < 10; i++ { ... }`
  - `for index, value := range collection { ... }`
  - `for condition { ... }`: Acts like a `while` loop.
- `switch`:

```
switch os := runtime.GOOS; os {
    case "darwin":
fmt.Println("macOS")
    case "linux": fmt.Println("Linux")
    default: fmt.Println("Others")
}
```

Functions and Methods

- Function:

```
func add(x int, y int) int {
    return x + y
}

// Multiple return values
func swap(x, y string) (string,
string) {
    return y, x
}
```

- Method (functions with a receiver):

```
type Circle struct { radius float64 }
func (c Circle) Area() float64 {
    return 3.14 * c.radius * c.radius
}
```

Structs and Interfaces

- Struct:

```
type Person struct {
    Name string
    Age  int
}
p := Person{Name: "Alice", Age: 30}
```

- Interface:

```
type Shaper interface {
    Area() float64
}
// Any type with an Area() method
implicitly satisfies this interface.
```

Pointers

- `&x`: Get the memory address of x.
- `*ptr`: Access the value at the address (dereferencing).
- Go does not support pointer arithmetic.

Concurrency (Goroutines and Channels)

- Goroutine: `go functionName()` (Runs the function asynchronously).
- Channel:
  - `ch := make(chan int)`
  - `ch <- 1`: Send data to the channel.
  - `val := <-ch`: Receive data from the channel.
  - `select`: Wait for multiple channel operations.

Error Handling

Go uses explicit value returns for error handling instead of exceptions.

```
val, err := doSomething()
if err != nil {
    // Handle error
    return
}
```

Defer, Panic, and Recover

- `defer`: Delay function execution until the surrounding function returns (mainly for resource cleanup).
- `panic`: Stop normal execution and start panicking.
- `recover`: Regain control of a panicking goroutine.

Google Style Guide

Core Principles

- Clarity: The intent and rationale of the code must be clear.
- Simplicity: Achieve goals in the simplest way possible.
- Concision: Increase the signal-to-noise ratio.
- Maintainability: Write code that is easy to maintain.
- Consistency: Maintain consistency with the entire codebase.

Naming Conventions

- MixedCaps: Use `MixedCaps` or `mixedCaps` (avoid underscores).
- Exported: Names starting with an uppercase letter are exported (public); others are package-private.
- Package: Short, clear, lowercase names (no underscores/caps).
- Receiver: Recommended 1-2 character short names (`f *File`).
- Variables: Use shorter names for narrower scopes.

Formatting

- gofmt: All Go files must be formatted with the `gofmt` tool.
- Line Length: No fixed limit, but consider refactoring if too long.
- Indentation: Use Tabs for indentation.

Programming Practices

- Error Handling: Follow the `if err != nil { return err }` pattern. Treat errors as values.
- Panic: Forbid `panic` in library code (recommend returning errors).
- Interfaces: Keep them small (usually 1-2 methods). Prefer defining them at the use site.

- Concurrency: Choose between channels and mutexes as appropriate. Channels are better for ownership and flow control.
- Tests: Recommend Table-driven testing.