Go Programming Comments Guide

Go uses two types of comments for documenting code and providing explanations.

Single-line Comments

Use `//` for single-line comments:

```go
// This is a single-line comment
var name string // This is also a comment

// You can stack multiple single-line comments
// to create multi-line explanations
func main() {
    fmt.Println("Hello") // Print greeting
}
```

Multi-line Comments

Use `/* */` for multi-line comments:

```go
/*
This is a multi-line comment
that spans several lines
*/

func calculate() {
    /*
    This function performs complex calculations
    using advanced algorithms
    */
    return 42
}
```

Best Practices

Package Comments

Document packages with a comment directly before the `package` statement:

```go
// Package math provides basic mathematical operations
// for financial calculations.
package math
```

Function Comments

Document exported functions with comments starting with the function name:

```go
// Add returns the sum of a and b.
func Add(a, b int) int {
    return a + b
}

// ProcessData validates and transforms the input data according to
// business rules before storing it in the database.
func ProcessData(data []byte) error {
    // implementation here
    return nil
}
```

Struct and Field Comments

Document types and their fields:

```go
// User represents a system user with authentication details.
type User struct {
    Name  string // User's display name
    Email string // Primary email address
    ID    int    // Unique identifier
}

// Config holds application configuration parameters.
type Config struct {
    Port     int    // Server port number
    Database string // Database connection string
    Debug    bool   // Enable debug logging
}
```

Variable and Constant Comments

Document important variables and constants:

```go
// DefaultTimeout is the maximum time to wait for a response.
const DefaultTimeout = 30 * time.Second

// userCache stores frequently accessed user data to improve performance.
var userCache = make(map[int]*User)
```

Comment Guidelines

DO:

- Start comments with the name of the item being documented
- Use complete sentences with proper punctuation
- Explain *why* something is done, not just *what* is done
- Keep comments concise but informative
- Update comments when code changes

DON'T:

- Comment obvious code:

```go
// Bad
i++ // increment i

// Good
i++ // move to next item in processing queue
```

- Use comments to explain bad code (refactor instead):

go

```go
// Bad
// This is a hack to work around the API limitation
result = strings.Replace(data, "bad", "good", -1)

// Better - refactor the code to be clearer
result = sanitizeAPIResponse(data)
```

Special Comment Types

TODO Comments

Mark future improvements or known issues:

go

```go
// TODO: Implement caching to improve performance
// TODO(username): Add input validation for edge cases
```

Build Tags

Use comments for build constraints:

go

```go
//go:build linux
// +build linux

package platform
```

Generated Code

Mark generated files:

go

```go
// Code generated by protoc-gen-go. DO NOT EDIT.
// source: user.proto
```

Documentation Tools

Go's built-in documentation tools (`go doc`, `godoc`, and `pkg.go.dev`) automatically generate documentation from properly formatted comments. Following these conventions ensures your code

documentation is accessible through these tools:

```bash
# View documentation for a package
go doc package-name

# View documentation for a specific function
go doc package-name.FunctionName

# Start local documentation server
godoc -http=:6060
```

Remember: Good comments explain the reasoning behind the code, not just what the code does. They help other developers (including your future self) understand the intent and context of your implementation.