Go Naming Conventions and Rules

General Principles

Go follows simple, consistent naming conventions that emphasize clarity and readability. The primary rule is that names should be **clear, concise, and meaningful**.

Case Sensitivity and Visibility

Public vs Private

- **Public** (exported): Names starting with an **uppercase letter**
- **Private** (unexported): Names starting with a **lowercase letter**

```go
// Public - accessible from other packages
var PublicVariable int
func PublicFunction() {}
type PublicStruct struct {}

// Private - only accessible within the same package
var privateVariable int
func privateFunction() {}
type privateStruct struct {}
```

Package Names

Rules

- **All lowercase**
- **Short and descriptive**
- **No underscores or mixed caps**
- Should match the directory name
- Avoid common names like `util`, `common`, `base`

Examples

go

```go
// Good
package http
package json
package strings
package fmt

// Bad
package HTTP
package JSON
package string_utils
package stringUtils
```

Variable Names

Local Variables

- Use **camelCase**
- Short names for short scopes
- Longer, descriptive names for longer scopes

go

```go
// Good - short scope
for i := 0; i < len(items); i++ {
    // i is fine here
}

// Good - longer scope
func processUserData() {
    userAccountBalance := getUserBalance()
    // descriptive name for longer-lived variable
}
```

Common Short Names

```go
// Conventional short names
i, j, k    // loop indices
n          // count
s          // string
r          // reader
w          // writer
c          // channel
ctx        // context
err        // error
```

Function Names

Rules

- Use **camelCase** for private functions
- Use **PascalCase** for public functions
- Avoid redundant words
- Use verbs for actions

```go
// Good
func getUserByID(id int) User {}
func validateEmail(email string) bool {}
func Start() {}
func Stop() {}

// Bad
func get_user_by_id(id int) User {}      // underscores
func GetUserByIDFunction(id int) User {} // redundant "Function"
func userByID(id int) User {}            // missing verb
```

Getters and Setters

```go
// Good - omit "Get" prefix for getters
func (u *User) Name() string { return u.name }
func (u *User) SetName(name string) { u.name = name }

// Bad
func (u *User) GetName() string { return u.name }
```

Type Names

Structs

- Use **PascalCase** for public structs
- Use **camelCase** for private structs
- Use singular nouns
- Avoid stuttering with package names

```go
// Good
type User struct {}
type HTTPClient struct {}

// In package 'user'
type Manager struct {} // Not UserManager

// Bad
type Users struct {}        // plural
type userHTTPClient struct {} // mixed case for private
```

Interfaces

- Often end with `-er` suffix
- Use single-word names when possible
- Describe behavior, not data

```go
// Good
type Reader interface {
    Read([]byte) (int, error)
}

type Writer interface {
    Write([]byte) (int, error)
}

type Stringer interface {
    String() string
}

// Bad
type ReaderInterface interface {} // redundant "Interface"
type IReader interface {}         // Hungarian notation
```

Constants

Rules

- Use **PascalCase** for public constants
- Use **camelCase** for private constants
- Group related constants in blocks
- Use meaningful names, not just numbers

go

```go
// Good
const (
    MaxRetries = 3
    DefaultTimeout = 30 * time.Second
)

const (
    StatusPending = iota
    StatusProcessing
    StatusComplete
    StatusFailed
)

// Private constants
const (
    bufferSize = 1024
    maxConnections = 100
)

// Bad
const MAX_RETRIES = 3           // underscores
const Default_Timeout = 30      // mixed case
const THREE = 3                 // meaningless name
```

Method Names

Rules

- Follow same rules as functions
- Receiver names should be short and consistent
- Use the same receiver name throughout a type's methods

```go
type User struct {
    name string
    age  int
}

// Good - consistent receiver name 'u'
func (u *User) Name() string { return u.name }
func (u *User) SetAge(age int) { u.age = age }
func (u *User) IsAdult() bool { return u.age >= 18 }

// Bad - inconsistent receiver names
func (user *User) Name() string { return user.name }
func (u *User) SetAge(age int) { u.age = age }
func (usr *User) IsAdult() bool { return usr.age >= 18 }
```

Channel Names

```go
// Good
done := make(chan bool)
results := make(chan Result)
userCh := make(chan User)

// Common pattern for channels
ch := make(chan int)    // when type is obvious from context
```

Error Variables

Rules

- Prefix with `Err` for sentinel errors
- Use descriptive names

```go
// Good
var (
    ErrNotFound = errors.New("not found")
    ErrInvalidInput = errors.New("invalid input")
    ErrTimeout = errors.New("operation timed out")
)

// In functions
if err != nil {
    return fmt.Errorf("failed to process user: %w", err)
}
```

File Names

Rules

- **All lowercase**
- Use **underscores** to separate words (unlike other Go names)
- Match the primary type or functionality

```go
// Good file names
user.go
http_client.go
string_utils.go
main.go

// Bad
User.go
httpClient.go
StringUtils.go
```

Test Names

Rules

- Test functions start with `Test`
- Benchmark functions start with `Benchmark`
- Example functions start with `Example`

```go
// Good
func TestUserValidation(t *testing.T) {}
func TestUserValidation_InvalidEmail(t *testing.T) {}
func BenchmarkStringConcat(b *testing.B) {}
func ExampleUser_String() {}

// Bad
func testUserValidation(t *testing.T) {}  // lowercase
func UserValidationTest(t *testing.T) {}  // wrong prefix
```

Common Patterns and Idioms

Acronyms and Initialisms

- Keep them uppercase in public names
- Lowercase in private names

```go
// Good
type HTTPClient struct {}
type URLParser struct {}
func ParseHTML() {}

// Private
type httpClient struct {}
func parseHTML() {}
```

Avoid Stuttering

```go
// Good
user.Manager    // not user.UserManager
log.Logger      // not log.LogLogger

// Bad
user.UserManager
log.LogLogger
```

Context Variables

```go
// Always use 'ctx' for context.Context
func processRequest(ctx context.Context, req Request) error {
    // ...
}
```

Documentation Comments

Rules

- Start with the name being documented
- Use complete sentences
- No extra formatting needed

```go
// User represents a system user with authentication credentials.
type User struct {
    name string
    id   int
}

// Name returns the user's display name.
func (u *User) Name() string {
    return u.name
}

// ProcessUsers handles batch user operations and returns
// the number of users successfully processed.
func ProcessUsers(users []User) (int, error) {
    // ...
}
```

Summary Checklist

- ✅ Use camelCase for private, PascalCase for public
- ✅ Keep package names short and lowercase
- ✅ Use meaningful variable names
- ✅ Omit "Get" from getter methods
- ✅ Use consistent receiver names

- ✅ Prefix error variables with "Err"
- ✅ Use underscores in file names only
- ✅ Avoid stuttering with package names
- ✅ Keep acronyms uppercase in public names
- ✅ Document public APIs starting with the name