What's in a name?
That which we call a rose
by any other name would
smell as sweet.

—*Romeo & Juliet*
Act 2 Scene 2

Dependency management?

Error values vs exceptions?

Generic programming?

“What’s with all the short variable names?!?”

“Readability is the defining quality of good code. Good names are critical to readability.”

–Andrew Gerrand

Formal parameters, return values, constants, functions, types, methods, file names, and packages

These are all identifiers we can declare

Choose identifiers for clarity, not brevity

“Every programmer has a variable naming philosophy. This is mine: A name's length should not exceed its information content.”

–Russ Cox

Local variables, formal parameters and return values, struct fields, and package level

You can declare many kinds of variables

The greater distance between declaration and use, the larger the identifer.

“Never use a long word
where a short one will do.”

–George Orwell
Politics and the English language

Lengthy bureaucratic names need to justify themselves

Names are contextual

```
for index := 0; index < len(s); index++ {
        //
}
```

```
for i := 0; i < len(s); i++ {
        //
}
```

```go
type Person struct {
        Name string
        Age  int
}

// AverageAge returns the average age of people.
func AverageAge(people []Person) int {
        if len(people) == 0 {
                return 0
        }

        var count, sum int
        for _, p := range people {
                sum += p.Age
                count += 1
        }

        return sum / count
}
```

Keep your friends close and your declarations closer

A variable's name should describe its contents, not its type

```
var usersMap      map[string]*User
var companiesMap  map[string]*Company
var productsMap   map[string]*Products
```

```
var (
        users     map[string]*User
        companies map[string]*Company
        products  map[string]*Products
)
```

Use a predictable naming style

“Choose variable names
that won’t be confused.”

—*Kernighan and Plauger*
Elements of Programming Style

```
func Query(d *sql.DB)

var dbase *sql.DB

type Result struct {
        DB *sql.DB
}

return func() (database *sql.DB, err error) { ... }
```

```
var db *sql.DB
```

`i`, `j`, and `k` are commonly the loop induction variable for simple `for` loops.

`n` is commonly associated with a counter or accumulator.

`v` is a common shorthand for a value in a generic encoding function, `k` is commonly used for the key of a map.

`a` and `b` are generic names for parameters comparing two variables of the same type.

`x` and `y` are generic names for local variables created for comparision

`s` is often used as shorthand for parameters of type `string` who's contents are opaque.

Collections; maps, slices, and arrays, should be pluralised.

Function names

“If a function is hard to name, maybe you’re giving the function too much responsibility.”

–Mike Kerr

Functions should be named for the result they return

“If you don’t know what a thing should be called, you cannot know what it is. If you don’t know what it is, you cannot sit down and write the code.”

–*Sam Gardiner*

```
func Add(a, b int) int
func Sum(a, b int) int
```

```
result := Add(37, 9)
result = Sum(37, 9)
```

```
func Maximum(a, b int) int
```

```
package grpc

func NewClient() *Client

func NewClientWithTimeout(timeout time.Duration) *Client
```

```go
type Option func(*Client) *Client

func NewClient(opts ...Option)

func WithTimeout(timeout time.Duration) func(c *Client)

client := grpc.NewClient(grpc.WithTimeout(10 * time.Second))
```

What about methods?

```
type BigDecimal struct {
    dollars, cents int
}

func (d *BigDecimal) Add(dollars, cents int)
func (d *BigDecimal) Sum(dollars, cents int)
```

```
var total BigDecimal
total.Sum(20, 5)
total.Add(9, 99)
```

A package's name should describe its purpose

“A package's name provides context for its contents, making it easier for clients to understand what the package is for and how to use it. […] Well-named packages make it easier to find the code you need.”

—*Sameer Ajmani*

An identifier's name includes the name of its package

The `Get` function from the `net/http` package becomes `http.Get` when referenced by another package.

The `Reader` type from the `strings` package becomes `strings.Reader` when imported into other packages.

The `net.Error` interface from the `net` package is clearly related to network errors.

Avoid package names like `base`, `common`, or `util`

“[A little] duplication is far cheaper than the wrong abstraction.”

—*Sandy Metz*

Resist the desire to create a package taxonomy

“The biggest issue Go developers have with application layout is thinking of packages as groups instead of layers.”

—*Ben Johnson*

Do not name your package v2 

```
import "github.com/pkg/term/v2" // bad

import "github.com/pkg/v2/term" // better
```

Don't let a package steal good variable names

```
func WriteLog(context context.Context, message string)
```

```go
func WriteLog(ctx context.Context, message string)
```

Conclusion

“Use the shortest name that carries the right amount of information in its context.”

–David Crawshaw

Brevity
A good name is concise. It carries a high signal to noise ratio.

Precision
A good name accurately describes the thing it represents.

Consistency
A good name should be predictable.

A variable's name should describe its contents

Use the smallest scope possible, declare variables close to their use.

Short variable names work well when the distance between their declaration and *last* use is short.

Prefer single letter variables for loops and branches, single words for parameters and return values, multiple words for functions and package level declarations.

Repeating the type of the variable in its name does not make it more type safe.

Functions, methods, and interfaces

Methods mutate state, functions transform data. Name them appropriately.

Functions should be named for the result they return.

Methods should be named for the action they perform.

Be wary of conjunctions, they could indicate a single function or method is doing too much.

A package's name should describe its purpose

Name your packages for what they provide, *not* what they contain.

Don’t create package taxonomies

There are two parts to each exported identifer, the identifiers name and its packages name, make use of that fact.

Package level variables deserve longer identifiers than locally scoped ones because their scope encompases the entire program.

Don't blow common identifiers on a package’s name.

Thank you!

Thank you for coming to Go Get Community