WIKIPEDIA

Go (programming language)

Go is a statically typed, compiled programming language designed at Google[10] by Robert Griesemer, Rob Pike, and Ken Thompson.[11] Go is syntactically similar to C, but with memory safety, garbage collection, structural typing,[5] and CSP-style concurrency.[12] The language is often referred to as **Golang** because of its former domain name, `golang.org`, but the proper name is Go.[13]

There are two major implementations:

- Google's self-hosting[14] "gc" compiler toolchain targeting multiple operating systems, and WebAssembly.[15]
- gofrontend, a frontend to other compilers, with the *libgo* library. With GCC the combination is gccgo;[16] with LLVM the combination is gollvm.[17][a]

A third-party source-to-source compiler, GopherJS,[19] compiles Go to JavaScript for front-end web development.

Go

Paradigm	Multi-paradigm: concurrent imperative, object-oriented[1][2]
Designed by	Robert Griesemer Rob Pike Ken Thompson
Developer	The Go Authors[3]
First appeared	November 10, 2009
Stable release	1.17.4[4] / 2 December 2021
Typing discipline	Inferred, static, strong, structural,[5][6] nominal
Implementation language	Go, Assembly language (gc); C++ (gofrontend)
OS	DragonFly BSD, FreeBSD, Linux, macOS, NetBSD, OpenBSD,[7] Plan 9,[8] Solaris, Windows
License	3-clause BSD[3] + patent grant[9]
Filename extensions	.go
Website	go.dev (https://go.dev)

Major implementations

gc, gofrontend

Contents

History
Version history
Design
Syntax
Types
Interface system
Package system
Concurrency: goroutines and channels
Suitability for parallel programming
Lack of race condition safety
Binaries
Omissions
Style
Tools
Examples
Hello world
Concurrency
Testing
Web App
Applications
Reception

Naming dispute
Criticism
See also
Notes
References
Further reading
External links

Influenced by
C, Oberon-2, Limbo, Active Oberon, communicating sequential processes, Pascal, Oberon, Smalltalk, Newsqueak, Modula-2, Alef, APL, BCPL, Modula, occam,Erlang,
Influenced
Odin, Crystal, Zig

History

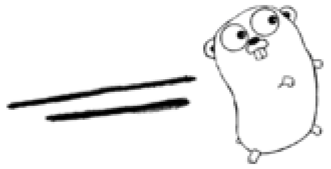
Go was designed at Google in 2007 to improve programming productivity in an era of multicore, networked machines and large codebases.[20] The designers wanted to address criticism of other languages in use at Google, but keep their useful characteristics:[21]

- static typing and run-time efficiency (like C),
- readability and usability (like Python or JavaScript),[22]
- high-performance networking and multiprocessing.

The designers were primarily motivated by their shared dislike of C++.[23][24][25]

Go was publicly announced in November 2009,[26] and version 1.0 was released in March 2012.[27][28] Go is widely used in production at Google[29] and in many other organizations and open-source projects.

In November 2016, the Go and Go Mono fonts were released by type designers Charles Bigelow and Kris Holmes specifically for use by the Go project. Go is a humanist sans-serif which resembles Lucida Grande and Go Mono is monospaced. Each of the fonts adhere to the WGL4 character set and were designed to be legible with a large x-height and distinct letterforms. Both Go and Go Mono adhere to the DIN 1450 standard by having a slashed zero, lowercase `l` with a tail, and an uppercase `I` with serifs.[30][31]


Mascot of Go programming language is a Gopher shown above.

In April 2018, the original logo was replaced with a stylized GO slanting right with trailing streamlines. However, the Gopher mascot remained the same.[32]

In August 2018, the Go principal contributors published two "draft designs" for new and incompatible "Go 2" language features, generics and error handling, and asked Go users to submit feedback on them.[33][34] Lack of support for generic programming and the verbosity of error handling in Go 1.x had drawn considerable criticism.

Version history

Go 1 guarantees compatibility[35] for the language specification and major parts of the standard library. All versions up to the current Go 1.17 release[36] have maintained this promise.

Each major Go release is supported until there are two newer major releases.[37]

Version history of Go

Major version	Initial release date	Language changes[38]	Other changes
1–1.0.3	2012-03-28	Initial release	
1.1–1.1.2	2013-05-13	▪ In Go 1.1, an integer division by constant zero is not a legal program, so it is a compile-time error. ▪ The definition of string and rune literals has been refined to exclude surrogate halves from the set of valid Unicode code points. ▪ Loosened return requirements rules. If the compiler can prove that a function always returns before reaching the end of a function, a final terminating statement can be omitted.	▪ The language allows the implementation to choose whether the `int` type and `uint` types are 32 or 64 bits. ▪ On 64-bit architectures, the maximum heap size has been enlarged substantially, from a few gigabytes to several tens of gigabytes. ▪ Addition of a race detector to the standard tool set.
1.2–1.2.2	2013-12-01	▪ The language now specifies that, for safety reasons, certain uses of nil pointers are guaranteed to trigger a run-time panic. ▪ Go 1.2 adds the ability to specify the capacity as well as the length when using a slicing operation on an existing array or slice. A slicing operation creates a new slice by describing a contiguous section of an already-created array or slice.	▪ The runtime scheduler can now be invoked on (non-inlined) function calls. ▪ Go 1.2 introduces a configurable limit (default 10,000) to the total number of threads a single program may have. ▪ In Go 1.2, the minimum size of the stack when a goroutine is created has been lifted from 4KB to 8KB.
1.3–1.3.3	2014-06-18	There are no language changes in this release.	▪ The Go 1.3 memory model adds a new rule concerning sending and receiving on buffered channels, to make explicit that a buffered channel can be used as a simple semaphore, using a send into the channel to acquire and a receive from the channel to release. ▪ Go 1.3 has changed the implementation of goroutine stacks away from the old, "segmented" model to a contiguous model. ▪ For a while now, the garbage collector has been *precise* when examining values in the heap; the Go 1.3 release adds equivalent precision to values on the stack. ▪ Iterations over small maps no longer happen in a consistent order. This is due to developers abusing implementation behaviour.
1.4–1.4.3	2014-12-10	▪ Range-expression without assignment ▪ Automatic double-dereference on method calls is now disallowed in gc and gccgo. This is a backwards incompatible change, but in	▪ In 1.4, much of the runtime code has been translated to Go so that the garbage collector can scan the stacks of programs in the runtime and get accurate information about what variables are active. ▪ The language accepted by the assemblers `cmd/5a`, `cmd/6a` and `cmd/8a` has had several changes,

		line with the language specification.	mostly to make it easier to deliver type information to the runtime. ▪ Addition of internal packages. ▪ New subcommand go generate.
1.5–1.5.4	2015-08-19	Due to an oversight, the rule that allowed the element type to be elided from slice literals was not applied to map keys. This has been corrected in Go 1.5.	▪ The compiler and runtime are now implemented in Go and assembler, without C. Now that the Go compiler and runtime are implemented in Go, a Go compiler must be available to compile the distribution from source. The compiler is now self-hosted. ▪ The garbage collector has been re-engineered for 1.5. The "stop the world" phase of the collector will almost always be under 10 milliseconds and usually much less. ▪ In Go 1.5, the order in which goroutines are scheduled has been changed.
1.6–1.6.4	2016-02-17	There are no language changes in this release.	▪ A major change was made to cgo defining the rules for sharing Go pointers with C code, to ensure that such C code can coexist with Go's garbage collector. ▪ The Go parser is now hand-written instead of generated. ▪ The `go vet` command now diagnoses passing function or method values as arguments to `Printf`, such as when passing `f` where `f()` was intended.
1.7–1.7.6	2016-08-15	Clarification on terminating statements in the language specification. This does not change existing behaviour.	▪ For 64-bit x86 systems, the following instructions have been added (see SSE): `PCMPESTRI`, `RORXL`, `RORXQ`, `VINSERTI128`, `VPADDD`, `VPADDQ`, `VPALIGNR`, `VPBLENDD`, `VPERM2F128`, `VPERM2I128`, `VPOR`, `VPSHUFB`, `VPSHUFD`, `VPSLLD`, `VPSLLDQ`, `VPSLLQ`, `VPSRLD`, `VPSRLDQ`, and `VPSRLQ` . ▪ This release includes a new code generation back end for 64-bit x86 systems, based on SSA. ▪ Packages using cgo may now include Fortran source files (in addition to C, C++, Objective C, and SWIG), although the Go bindings must still use C language APIs. ▪ The new subcommand "`go tool dist list`" prints all supported operating system/architecture pairs.
1.8–1.8.7	2017-02-16	When explicitly converting a value from one struct type to another, as of Go 1.8 the tags are ignored. Thus two structs that differ only in their tags may be converted from one to the other.	▪ For 64-bit x86 systems, the following instructions have been added: `VBROADCASTSD`, `BROADCASTSS`, `MOVDDUP`, `MOVSHDUP`, `MOVSLDUP`, `VMOVDDUP`, `VMOVSHDUP`, and `VMOVSLDUP`. ▪ Garbage collection pauses should be significantly shorter than they were in Go 1.7, usually under 100 microseconds and often as low as 10 microseconds. See the document on eliminating stop-the-world stack re-scanning for details. ▪ The overhead of deferred function calls has been reduced by about half. ▪ The overhead of calls from Go into C has been reduced by about half.
1.9–1.9.7	2017-08-24	▪ Go now supports type aliases. ▪ Force the intermediate rounding in floating-point arithmetic.	The Go compiler now supports compiling a package's functions in parallel, taking advantage of multiple cores.

1.10–1.10.7	2018-02-16	▪ A corner case involving shifts of untyped constants has been clarified. ▪ The grammar for method expressions has been updated to relax the syntax to allow any type expression as a receiver.	For the x86 64-bit port, the assembler now supports 359 new instructions, including the full AVX, AVX2, BMI, BMI2, F16C, FMA3, SSE2, SSE3, SSSE3, SSE4.1, and SSE4.2 extension sets. The assembler also no longer implements `MOVL $0, AX` as an `XORL` instruction, to avoid clearing the condition flags unexpectedly.
1.11–1.11.6	2018-08-24	There are no language changes in this release.	▪ Go 1.11 adds an experimental port to WebAssembly. ▪ Go 1.11 adds preliminary support for a new concept called "modules", an alternative to GOPATH with integrated support for versioning and package distribution. ▪ The assembler for `amd64` now accepts AVX512 instructions. ▪ Go 1.11 drops support of Windows XP and Windows Vista.[39] ▪ Go 1.11.3 and later fix the TLS authentication vulnerability in the crypto/x509 package.[40]
1.12.1	2019-02-25	There are no language changes in this release.	▪ Opt-in support for TLS 1.3 ▪ Improved modules support (in preparation for being the default in Go 1.13) ▪ Support for `windows/arm` ▪ Improved macOS & iOS forwards compatibility
1.13.1	2019-09-03	Go now supports a more uniform and modernized set of number literal prefixes	▪ support for TLS 1.3 in the crypto/tls package by default (opt-out will be removed in Go 1.14) ▪ Support for error wrapping[41]
1.14	2020-02-25	Permits embedding interfaces with overlapping method sets[42]	Module support in the `go` command is now ready for production use[42]
1.15	2020-08-11	There are no language changes in this release.	▪ New embedded time/tzdata package[43] ▪ The printing verbs `%#g` and `%#G` now preserve trailing zeros for floating-point values[44] ▪ Package `reflect` now disallows accessing methods of all non-exported fields, whereas previously it allowed accessing those of non-exported, embedded fields.
1.16	2021-02-16	There are no language changes in this release.	▪ New support for embedding files inside a go program ▪ Support for macos/arm ▪ Module-aware mode is enabled by default[45]
1.17	2021-08-16	Slices may now be converted to array pointers pointing to the same data[46]	▪ Go 1.17 now passes function arguments in registers instead of on the stack (only on amd64), giving a 5% performance boost on average[46]

Design

Go is influenced by C (especially the Plan 9 dialect), but with an emphasis on greater simplicity and safety. The language consists of:

- A syntax and environment adopting patterns more common in dynamic languages:[47]
  - Optional concise variable declaration and initialization through type inference (`x := 0` instead of `int x = 0;` or `var x = 0;`).
  - Fast compilation.[48]
  - Remote package management (`go get`)[49] and online package documentation.[50]
- Distinctive approaches to particular problems:
  - Built-in concurrency primitives: light-weight processes (goroutines), channels, and the `select` statement.
  - An interface system in place of virtual inheritance, and type embedding instead of non-virtual inheritance.
  - A toolchain that, by default, produces statically linked native binaries without external dependencies.
- A desire to keep the language specification simple enough to hold in a programmer's head,[51] in part by omitting features that are common in similar languages.

Syntax

Go's syntax includes changes from C aimed at keeping code concise and readable. A combined declaration/initialization operator was introduced that allows the programmer to write `i := 3` or `s := "Hello, world!"`, without specifying the types of variables used. This contrasts with C's `int i = 3;` and `const char *s = "Hello, world!";`. Semicolons still terminate statements,[b] but are implicit when the end of a line occurs.[c] Methods may return multiple values, and returning a `result, err` pair is the conventional way a method indicates an error to its caller in Go.[d] Go adds literal syntaxes for initializing struct parameters by name and for initializing maps and slices. As an alternative to C's three-statement `for` loop, Go's `range` expressions allow concise iteration over arrays, slices, strings, maps, and channels.[54]

Types

Go has a number of built-in types, including numeric ones (`byte`, `int64`, `float32`, etc.), booleans, and character strings (`string`). Strings are immutable; built-in operators and keywords (rather than functions) provide concatenation, comparison, and UTF-8 encoding/decoding.[55] Record types can be defined with the `struct` keyword.[56]

For each type *T* and each non-negative integer constant *n*, there is an array type denoted `[`*n*`]`*T*; arrays of differing lengths are thus of different types. Dynamic arrays are available as "slices", denoted `[]`*T* for some type *T*. These have a length and a *capacity* specifying when new memory needs to be allocated to expand the array. Several slices may share their underlying memory.[57][58][59]

Pointers are available for all types, and the pointer-to-*T* type is denoted `*`*T*. Address-taking and indirection use the `&` and `*` operators, as in C, or happen implicitly through the method call or attribute access syntax.[60][61] There is no pointer arithmetic,[e] except via the special `unsafe.Pointer` type in the standard library.[62]

For a pair of types *K*, *V*, the type `map[`*K*`]`*V* is the type of hash tables mapping type-*K* keys to type-*V* values. Hash tables are built into the language, with special syntax and built-in functions. `chan` *T* is a *channel* that allows sending values of type *T* between concurrent Go processes.

Aside from its support for interfaces, Go's type system is nominal: the `type` keyword can be used to define a new *named type*, which is distinct from other named types that have the same layout (in the case of a `struct`, the same members in the same order). Some conversions between types (e.g., between the various integer types) are pre-defined and adding a new type may define additional conversions, but conversions between named types must always be invoked explicitly.[63] For example, the `type` keyword can be used to define a type for IPv4 addresses, based on 32-bit unsigned integers:

```go
type ipv4addr uint32
```

With this type definition, `ipv4addr(x)` interprets the `uint32` value `x` as an IP address. Simply assigning `x` to a variable of type `ipv4addr` is a type error.

Constant expressions may be either typed or "untyped"; they are given a type when assigned to a typed variable if the value they represent passes a compile-time check.[64]

Function types are indicated by the `func` keyword; they take zero or more parameters and return zero or more values, all of which are typed. The parameter and return values determine a function type; thus, `func(string, int32) (int, error)` is the type of functions that take a `string` and a 32-bit signed integer, and return a signed integer (of default width) and a value of the built-in interface type `error`.

Any named type has a method set associated with it. The IP address example above can be extended with a method for checking whether its value is a known standard:

```go
// ZeroBroadcast reports whether addr is 255.255.255.255.
func (addr ipv4addr) ZeroBroadcast() bool {
    return addr == 0xFFFFFFFF
}
```

Due to nominal typing, this method definition adds a method to `ipv4addr`, but not on `uint32`. While methods have special definition and call syntax, there is no distinct method type.[65]

Interface system

Go provides two features that replace class inheritance.

The first is *embedding*, which can be viewed as an automated form of composition[66] or delegation.[67]:255

The second are its *interfaces*, which provides runtime polymorphism.[68]:266 Interfaces are a class of types and provide a limited form of structural typing in the otherwise nominal type system of Go. An object which is of an interface type is also of another type, much like C++ objects being simultaneously of a base and derived class. Go interfaces were designed after protocols from the Smalltalk programming language.[69] Multiple sources use the term duck typing when describing Go interfaces.[70][71] Although the term duck typing is not precisely defined and therefore not wrong, it usually implies that type conformance is not statically checked. Since conformance to a Go interface is checked statically by the Go compiler (except when performing a type assertion), the Go authors prefer the term *structural typing*.[72]

The definition of an interface type lists required methods by name and type. Any object of type T for which functions exist matching all the required methods of interface type I is an object of type I as well. The definition of type T need not (and cannot) identify type I. For example, if `Shape`, `Square` and `Circle` are defined as

```go
import "math"

type Shape interface {
    Area() float64
}

type Square struct { // Note: no "implements" declaration
    side float64
}

func (sq Square) Area() float64 { return sq.side * sq.side }

type Circle struct { // No "implements" declaration here either
    radius float64
}

func (c Circle) Area() float64 { return math.Pi * math.Pow(c.radius, 2) }
```

then both a `Square` and a `Circle` are implicitly a `Shape` and can be assigned to a `Shape`-typed variable.[68]:263–268 In formal language, Go's interface system provides structural rather than nominal typing. Interfaces can embed other interfaces with the effect of creating a combined interface that is satisfied by exactly the types that implement the embedded interface and any methods that the newly defined interface adds.[68]:270

The Go standard library uses interfaces to provide genericity in several places, including the input/output system that is based on the concepts of `Reader` and `Writer`.[68]:282–283

Besides calling methods via interfaces, Go allows converting interface values to other types with a run-time type check. The language constructs to do so are the *type assertion*,[73] which checks against a single potential type, and the *type switch*,[74] which checks against multiple types.

The *empty interface* `interface{}` is an important base case because it can refer to an item of *any* concrete type. It is similar to the `Object` class in Java or C# and is satisfied by any type, including built-in types like `int`.[68]:284 Code using the empty interface cannot simply call methods (or built-in operators) on the referred-to object, but it can store the `interface{}` value, try to convert it to a more useful type via a type assertion or type switch, or inspect it with Go's `reflect` package.[75] Because `interface{}` can refer to any value, it is a limited way to escape the restrictions of static typing, like `void*` in C but with additional run-time type checks.

The `interface{}` type can be used to model structured data of any arbitrary schema in Go, such as JSON or YAML data, by representing it as a `map[string]interface{}` (map of string to empty interface). This recursively describes data in the form of a dictionary with string keys and values of any type.[76]

Interface values are implemented using pointer to data and a second pointer to run-time type information.[77] Like some other types implemented using pointers in Go, interface values are `nil` if uninitialized.[78]

Package system

In Go's package system, each package has a path (e.g., `"compress/bzip2"` or `"golang.org/x/net/html"`) and a name (e.g., `bzip2` or `html`). References to other packages' definitions must *always* be prefixed with the other package's name, and only the *capitalized* names from other packages are accessible: `io.Reader` is public but `bzip2.reader` is not.[79] The `go get` command can retrieve packages stored in a remote repository[80] and developers are encouraged to develop packages inside a base path corresponding to a source repository (such as example.com/user_name/package_name) to reduce the likelihood of name collision with future additions to the standard library or other external libraries.[81]

Proposals exist to introduce a proper package management solution for Go similar to CPAN for Perl or Rust's cargo system or Node's npm system.[82]

Concurrency: goroutines and channels

The Go language has built-in facilities, as well as library support, for writing concurrent programs. Concurrency refers not only to CPU parallelism, but also to asynchrony: letting slow operations like a database or network read run while the program does other work, as is common in event-based servers.[83]

The primary concurrency construct is the *goroutine*, a type of light-weight process. A function call prefixed with the `go` keyword starts a function in a new goroutine. The language specification does not specify how goroutines should be implemented, but current implementations multiplex a Go process's goroutines onto a smaller set of operating-system threads, similar to the scheduling performed in Erlang.[84]:10

While a standard library package featuring most of the classical concurrency control structures (mutex locks, etc.) is available,[84]:151–152 idiomatic concurrent programs instead prefer *channels*, which provide send messages between goroutines.[85] Optional buffers store messages in FIFO order[67]:43 and allow sending goroutines to proceed before their messages are received.

Channels are typed, so that a channel of type `chan` *T* can only be used to transfer messages of type *T*. Special syntax is used to operate on them; `<-ch` is an expression that causes the executing goroutine to block until a value comes in over the channel `ch`, while `ch <- x` sends the value `x` (possibly blocking until another goroutine receives the value). The built-in `switch`-like `select` statement can be used to implement non-blocking communication on multiple channels; see below for an example. Go has a memory model describing how goroutines must use channels or other operations to safely share data.[86]

The existence of channels sets Go apart from actor model-style concurrent languages like Erlang, where messages are addressed directly to actors (corresponding to goroutines). The actor style can be simulated in Go by maintaining a one-to-one correspondence between goroutines and channels, but the language allows multiple goroutines to share a channel or a single goroutine to send and receive on multiple channels.[84]:147

From these tools one can build concurrent constructs like worker pools, pipelines (in which, say, a file is decompressed and parsed as it downloads), background calls with timeout, "fan-out" parallel calls to a set of services, and others.[87] Channels have also found uses further from the usual notion of interprocess communication, like serving as a concurrency-safe list of recycled buffers,[88] implementing coroutines (which helped inspire the name *goroutine*),[89] and implementing iterators.[90]

Concurrency-related structural conventions of Go (channels and alternative channel inputs) are derived from Tony Hoare's communicating sequential processes model. Unlike previous concurrent programming languages such as Occam or Limbo (a language on which Go co-designer

Rob Pike worked),[91] Go does not provide any built-in notion of safe or verifiable concurrency.[92] While the communicating-processes model is favored in Go, it is not the only one: all goroutines in a program share a single address space. This means that mutable objects and pointers can be shared between goroutines; see § Lack of race condition safety, below.

Suitability for parallel programming

Although Go's concurrency features are not aimed primarily at parallel processing,[83] they can be used to program shared-memory multi-processor machines. Various studies have been done into the effectiveness of this approach.[93] One of these studies compared the size (in lines of code) and speed of programs written by a seasoned programmer not familiar with the language and corrections to these programs by a Go expert (from Google's development team), doing the same for Chapel, Cilk and Intel TBB. The study found that the non-expert tended to write divide-and-conquer algorithms with one `go` statement per recursion, while the expert wrote distribute-work-synchronize programs using one goroutine per processor. The expert's programs were usually faster, but also longer.[94]

Lack of race condition safety

There are no restrictions on how goroutines access shared data, making race conditions possible. Specifically, unless a program explicitly synchronizes via channels or other means, writes from one goroutine might be partly, entirely, or not at all visible to another, often with no guarantees about ordering of writes.[92] Furthermore, Go's *internal data structures* like interface values, slice headers, hash tables, and string headers are not immune to race conditions, so type and memory safety can be violated in multithreaded programs that modify shared instances of those types without synchronization.[95][96] Instead of language support, safe concurrent programming thus relies on conventions; for example, Chisnall recommends an idiom called "aliases xor mutable", meaning that passing a mutable value (or pointer) over a channel signals a transfer of ownership over the value to its receiver.[84]:155

Binaries

The linker in the gc toolchain creates statically linked binaries by default; therefore all Go binaries include the Go runtime.[97][98]

Omissions

Go deliberately omits certain features common in other languages, including (implementation) inheritance, generic programming, assertions,[f] pointer arithmetic,[e] implicit type conversions, untagged unions,[g] and tagged unions.[h] The designers added only those facilities that all three agreed on.[101]

Of the omitted language features, the designers explicitly argue against assertions and pointer arithmetic, while defending the choice to omit type inheritance as giving a more useful language, encouraging instead the use of interfaces to achieve dynamic dispatch[i] and composition to reuse code. Composition and delegation are in fact largely automated by `struct` embedding; according to researchers Schmager *et al.*, this feature "has many of the drawbacks of inheritance: it affects the public interface of objects, it is not fine-grained (i.e, no method-level control over embedding), methods of embedded objects cannot be hidden, and it is static", making it "not obvious" whether programmers will overuse it to the extent that programmers in other languages are reputed to overuse inheritance.[66]

The designers express an openness to generic programming and note that built-in functions *are* in fact type-generic, but these are treated as special cases; Pike calls this a weakness that may at some point be changed.[57] The Google team built at least one compiler for an experimental Go dialect with generics, but did not release it.[102] They are also open to standardizing ways to apply code generation.[103] In June 2020, a new draft design document[104] was published, which would add the necessary syntax to Go for declaring generic functions and types. A code translation tool `go2go` was provided to allow users to try out the new syntax, along with a generics-enabled version of the online Go Playground.[105]

Initially omitted, the exception-like `panic`/`recover` mechanism was eventually added, which the Go authors advise using for unrecoverable errors such as those that should halt an entire program or server request, or as a shortcut to propagate errors up the stack within a package (but not across package boundaries; there, error returns are the standard API).[106][107][108][109]

Style

The Go authors put substantial effort into influencing the style of Go programs:

- Indentation, spacing, and other surface-level details of code are automatically standardized by the `gofmt` tool.[110] `golint` does additional style checks automatically.
- Tools and libraries distributed with Go suggest standard approaches to things like API documentation (`godoc`),[111] testing (`go test`), building (`go build`), package management (`go get`), and so on.
- Go enforces rules that are recommendations in other languages, for example banning cyclic dependencies, unused variables[112] or imports,[113] and implicit type conversions.
- The *omission* of certain features (for example, functional-programming shortcuts like `map` and Java-style `try`/`finally` blocks) tends to encourage a particular explicit, concrete, and imperative programming style.
- On day one the Go team published a collection of Go idioms,[111] and later also collected code review comments,[114] talks,[115] and official blog posts[116] to teach Go style and coding philosophy.

Tools

The main Go distribution includes tools for building, testing, and analyzing code:

- `go build`, which builds Go binaries using only information in the source files themselves, no separate makefiles
- `go test`, for unit testing and microbenchmarks
- `go fmt`, for formatting code
- `go install`, for retrieving and installing remote packages
- `go vet`, a static analyzer looking for potential errors in code
- `go run`, a shortcut for building and executing code
- `godoc`, for displaying documentation or serving it via HTTP
- `gorename`, for renaming variables, functions, and so on in a type-safe way
- `go generate`, a standard way to invoke code generators

It also includes profiling and debugging support, runtime instrumentation (for example, to track garbage collection pauses), and a race condition tester.

An ecosystem of third-party tools adds to the standard distribution, such as `gocode`, which enables code autocompletion in many text editors, `goimports`, which automatically adds/removes package imports as needed, and `errcheck`, which detects code that might unintentionally ignore errors.

Examples

Hello world

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, world!")
}
```

where "fmt" is the package for *formatted I/O*, similar to C's C file input/output.[117]

Concurrency

The following simple program demonstrates Go's concurrency features to implement an asynchronous program. It launches two lightweight threads ("goroutines"): one waits for the user to type some text, while the other implements a timeout. The `select` statement waits for either of these goroutines to send a message to the main routine, and acts on the first message to arrive (example adapted from David Chisnall's book).[84]:152

```
package main

import (
    "fmt"
    "time"
)

func readword(ch chan string) {
    fmt.Println("Type a word, then hit Enter.")
    var word string
    fmt.Scanf("%s", &word)
    ch <- word
}

func timeout(t chan bool) {
    time.Sleep(5 * time.Second)
    t <- false
}

func main() {
    t := make(chan bool)
    go timeout(t)

    ch := make(chan string)
    go readword(ch)

    select {
    case word := <-ch:
        fmt.Println("Received", word)
    case <-t:
        fmt.Println("Timeout.")
    }
}
```

Testing

The testing package provides support for automated testing of go packages.[118] Target function example:

```
func ExtractUsername(email string) string {
    at := strings.Index(email, "@")
    return email[:at]
}
```

Test code (note that **assert** keyword is missing in Go; tests live in <filename>_test.go at the same package):

```
import (
    "testing"
)

func TestExtractUsername(t *testing.T) {
    t.Run("withoutDot", func(t *testing.T) {
        username := ExtractUsername("r@google.com")
        if username != "r" {
            t.Fatalf("Got: %v\n", username)
        }
    })

    t.Run("withDot", func(t *testing.T) {
        username := ExtractUsername("jonh.smith@example.com")
        if username != "jonh.smith" {
            t.Fatalf("Got: %v\n", username)
        }
    })
}
```

It is possible to run tests in parallel.

Web App

The net/http package provides support for creating web applications.

This example would show "Hello world!" when localhost:8080 is visited.

```
package main

import (
    "fmt"
    "log"
    "net/http"
)

func helloFunc(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello world!")
}

func main() {
    http.HandleFunc("/", helloFunc)
    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

Applications

Some notable open-source applications written in Go include:[119]

- Caddy, an open source HTTP/2 web server with automatic HTTPS capability
- CockroachDB, an open source, survivable, strongly consistent, scale-out SQL database
- Consul, a software for DNS-based service discovery and providing distributed Key-value storage, segmentation and configuration.
- Docker, a set of tools for deploying Linux containers
- EdgeX, a vendor-neutral open-source platform hosted by the Linux Foundation, providing a common framework for industrial IoT edge computing[120]
- Hugo, a static site generator
- InfluxDB, an open source database specifically to handle time series data with high availability and high performance requirements
- InterPlanetary File System, a content-addressable, peer-to-peer hypermedia protocol[121]
- Juju, a service orchestration tool by Canonical, packagers of Ubuntu Linux
- Kubernetes container management system
- lnd, an implementation of the Bitcoin Lightning Network[122]
- Mattermost, a teamchat system
- NATS Messaging, an open-source messaging system featuring the core design principles of performance, scalability, and ease of use[123]
- OpenShift, a cloud computing platform as a service by Red Hat
- Rclone, a command line program to manage files on cloud storage and other high latency services
- Snappy, a package manager for Ubuntu Touch developed by Canonical
- Syncthing, an open-source file synchronization client/server application
- Terraform, an open-source, multiple cloud infrastructure provisioning tool from HashiCorp
- TiDB, an open-source, distributed HTAP database compatible with the MySQL protocol from PingCAP

Other notable companies and sites using Go (generally together with other languages, not exclusively) include:

- Cacoo, for their rendering of the user dashboard page and microservice using Go and gRPC[124]
- Chango, a programmatic advertising company uses Go in its real-time bidding systems[125]
- Cloud Foundry, a platform as a service[126]
- Cloudflare, for their delta-coding proxy Railgun, their distributed DNS service, as well as tools for cryptography, logging, stream processing, and accessing SPDY sites[127][128]
- Container Linux (formerly CoreOS), a Linux-based operating system that uses Docker containers[129] and rkt containers
- Couchbase, Query and Indexing services within the Couchbase Server[130]
- Dropbox, who migrated some of their critical components from Python to Go[131]
- Ethereum, The *go-ethereum* implementation of the Ethereum Virtual Machine blockchain for the *Ether* cryptocurrency[132]
- Gitlab, a web-based DevOps lifecycle tool that provides a Git-repository, wiki, issue-tracking, continuous integration, deployment pipeline features[133]
- Google, for many projects, notably including download server dl.google.com[134][135][136]
- Heroku, for Doozer, a lock service[12]
- Hyperledger Fabric, an open source, enterprise-focused distributed ledger project
- MongoDB, tools for administering MongoDB instances[137]
- Netflix, for two portions of their server architecture[138]

- Nutanix, for a variety of micro-services in its Enterprise Cloud OS[139]
- Plug.dj, an interactive online social music streaming website[140]
- SendGrid, a Boulder, Colorado-based transactional email delivery and management service.[141]
- SoundCloud, for "dozens of systems"[142]
- Splice, for the entire backend (API and parsers) of their online music collaboration platform[143]
- ThoughtWorks, some tools and applications for continuous delivery and instant messages (CoyIM)[144]
- Twitch, for their IRC-based chat system (migrated from Python)[145]
- Uber, for handling high volumes of geofence-based queries[146]

See also related query to Wikidata (https://query.wikidata.org/#SELECT%20DISTINCT%20%3Finstance_of%20%3Finstance_ofDescription%20%3Finstance_ofLabel%20%3Fofficial_website%0AWHERE%20%7B%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%2Cen%22.%20%7D%0A%20%20%3Finstance_of%20wdt%3AP31%2Fwdt%3AP279%2a%20wd%3AQ341%0A%20%20OPTIONAL%20%7B%20%3Finstance_of%20wdt%3AP856%20%3Fofficial_website%20%7D%0A%20%20%3Finstance_of%20wdt%3AP277%20wd%3AQ37227.%0A%7D).

Reception

The interface system, and the deliberate omission of inheritance, were praised by Michele Simionato, who likened these characteristics to those of Standard ML, calling it "a shame that no popular language has followed [this] particular route".[147]

Dave Astels at Engine Yard wrote:[148]

> Go is extremely easy to dive into. There are a minimal number of fundamental language concepts and the syntax is clean and designed to be clear and unambiguous. Go *is* still experimental and still a little rough around the edges.

Go was named Programming Language of the Year by the TIOBE Programming Community Index in its first year, 2009, for having a larger 12-month increase in popularity (in only 2 months, after its introduction in November) than any other language that year, and reached 13th place by January 2010,[149] surpassing established languages like Pascal. By June 2015, its ranking had dropped to below 50th in the index, placing it lower than COBOL and Fortran.[150] But as of January 2017, its ranking had surged to 13th, indicating significant growth in popularity and adoption. Go was awarded TIOBE programming language of the year 2016.

Bruce Eckel has stated:[151]

> The complexity of C++ (even more complexity has been added in the new C++), and the resulting impact on productivity, is no longer justified. All the hoops that the C++ programmer had to jump through in order to use a C-compatible language make no sense anymore -- they're just a waste of time and effort. Go makes much more sense for the class of problems that C++ was originally intended to solve.

A 2011 evaluation of the language and its `gc` implementation in comparison to C++ (GCC), Java and Scala by a Google engineer found:

> Go offers interesting language features, which also allow for a concise and standardized notation. The compilers for this language are still immature, which reflects in both performance and binary sizes.
>
> — R. Hundt[152]

The evaluation got a rebuttal from the Go development team. Ian Lance Taylor, who had improved the Go code for Hundt's paper, had not been aware of the intention to publish his code, and says that his version was "never intended to be an example of idiomatic or efficient Go"; Russ Cox then optimized the Go code, as well as the C++ code, and got the Go code to run slightly faster than C++ and more than an order of magnitude faster than the code in the paper.[153]

Naming dispute

On November 10, 2009, the day of the general release of the language, Francis McCabe, developer of the Go! programming language (note the exclamation point), requested a name change of Google's language to prevent confusion with his language, which he had spent 10 years developing.[154] McCabe raised concerns that "the 'big guy' will end up steam-rollering over" him, and this concern resonated with the more than 120 developers who commented on Google's official issues thread saying they should change the name, with some[155] even saying the issue contradicts Google's motto of: Don't be evil.[156]

On October 12, 2010, the issue was closed by Google developer Russ Cox (@rsc) with the custom status "Unfortunate" accompanied by the following comment:

> "There are many computing products and services named Go. In the 11 months since our release, there has been minimal confusion of the two languages."[156]

Criticism

Go critics say that:

- The lack of parametric polymorphism for generic programming leads to code duplication or unsafe type conversions and flow-disrupting verbosity.[157][158][159][160]
- Go's *nil* combined with the lack of algebraic types leads to difficulty handling failures and base cases.[157][159]
- Go does not allow an opening brace to appear on its own line, which forces all Go programmers to use the same brace style.[161]
- File semantics in Go standard library are heavily based on POSIX semantics, and they do not map well to the Windows platform.[162][163] Note that this problem is not particular to Go, but other programming languages have solved it through well defined standard libraries.

Study shows that it is as easy to make concurrency bugs with message passing as with shared memory, sometimes even more.[164]

See also

- Comparison of programming languages

Notes

a. Using alternative backends reduces compilation speed and Go's control over garbage collection but provides better machine-code optimization.[18]
b. But "To allow complex statements to occupy a single line, a semicolon may be omitted before a closing) or }".[52]
c. "if the newline comes after a token that could end a statement, [the lexer will] insert a semicolon".[53]
d. Usually, exactly one of the result and error values has a value other than the type's zero value; sometimes both do, as when a read or write can only be partially completed, and sometimes neither, as when a read returns 0 bytes. See Semipredicate problem: Multivalued return.
e. Language FAQ "Why is there no pointer arithmetic? Safety ... never derive an illegal address that succeeds incorrectly ... using array indices can be as efficient as ... pointer arithmetic ... simplify the implementation of the garbage collector...."[11]
f. Language FAQ "Why does Go not have assertions? ...our experience has been that programmers use them as a crutch to avoid thinking about proper error handling and reporting...."[11]
g. Language FAQ "Why are there no untagged unions...? [they] would violate Go's memory safety guarantees."[11]
h. Language FAQ "Why does Go not have variant types? ... We considered [them but] they overlap in confusing ways with interfaces.... [S]ome of what variant types address is already covered, ... although not as elegantly."[11] (The tag of an interface type[99] is accessed with a type assertion[100]).
i. Questions "How do I get dynamic dispatch of methods?" and "Why is there no type inheritance?" in the language FAQ.[11]

References

funkyprogrammer.uk/concurrency-in-go-programming-language/ (https://funkyprogrammer.uk/concurrency-in-go-programming-language/)

1. "Is Go an object-oriented language?" (https://golang.org/doc/faq#Is_Go_an_object-oriented_language). Retrieved April 13, 2019. "Although Go has types and methods and allows an object-oriented style of programming, there is no type hierarchy."
2. "Go: code that grows with grace" (https://talks.golang.org/2012/chat.slide#5). Retrieved June 24, 2018. "Go is Object Oriented, but not in the usual way."
3. "Text file LICENSE" (https://golang.org/LICENSE). *The Go Programming Language*. Retrieved October 5, 2012.
4. "Go 1.17" (https://go.dev/doc/devel/release#go1.17).
5. "Why doesn't Go have "implements" declarations?" (https://golang.org/doc/faq#implements_interface). *golang.org*. Retrieved October 1, 2015.
6. Pike, Rob (December 22, 2014). "Rob Pike on Twitter" (https://twitter.com/rob_pike/status/546973312543227904). Retrieved March 13, 2016. "Go has structural typing, not duck typing. Full interface satisfaction is checked and required."
7. "lang/go: go-1.4 – Go programming language" (http://ports.su/lang/go). *OpenBSD ports*. December 23, 2014. Retrieved January 19, 2015.
8. "Go Porting Efforts" (http://go-lang.cat-v.org/os-ports). *Go Language Resources*. cat-v. January 12, 2010. Retrieved January 18, 2010.

9. "Additional IP Rights Grant" (https://golang.org/PATENTS). *The Go Programming Language*. Retrieved October 5, 2012.
10. Kincaid, Jason (November 10, 2009). "Google's Go: A New Programming Language That's Python Meets C++" (https://techcrunch.com/2009/11/10/google-go-language/). *TechCrunch*. Retrieved January 18, 2010.
11. "Language Design FAQ" (https://golang.org/doc/go_faq.html). *golang.org*. January 16, 2010. Retrieved February 27, 2010.
12. Metz, Cade (May 5, 2011). "Google Go boldly goes where no code has gone before" (https://www.theregister.co.uk/2011/05/05/google_go/). *The Register*.
13. "Is the language called Go or Golang?" (https://tip.golang.org/doc/faq#go_or_golang). Retrieved March 26, 2020. "The language is called Go."
14. "Go 1.5 Release Notes" (https://golang.org/doc/go1.5#implementation). Retrieved January 28, 2016. "The compiler and runtime are now implemented in Go and assembler, without C."
15. "Go 1.11 is Released - The Go Blog" (https://blog.golang.org/go1.11). August 24, 2018. Retrieved January 1, 2019.
16. "Installing GCC: Configuration" (https://gcc.gnu.org/install/configure.html). Retrieved December 3, 2011. "Ada, Go and Objective-C++ are not default languages"
17. "FAQ: Implementation" (https://golang.org/doc/go_faq.html#Implementation). *golang.org*. August 2, 2021. Retrieved August 2, 2021.
18. "gollvm § Is gollvm a replacement for the main Go compiler? (gc)" (https://go.googlesource.com/gollvm/). *Git at Google*.
19. "A compiler from Go to JavaScript for running Go code in a browser: Gopherjs/Gopherjs" (https://github.com/gopherjs/gopherjs). April 18, 2020.
20. "Go at Google: Language Design in the Service of Software Engineering" (https://talks.golang.org/2012/splash.article). Retrieved October 8, 2018.
21. Pike, Rob (April 28, 2010). "Another Go at Language Design" (http://www.stanford.edu/class/ee380/Abstracts/100428.html). *Stanford EE Computer Systems Colloquium*. Stanford University. Video available (https://www.youtube.com/watch?v=7VcArS4Wpqk).
22. "Frequently Asked Questions (FAQ) - The Go Programming Language" (https://golang.org/doc/faq#different_syntax). *golang.org*. Retrieved February 26, 2016.
23. Binstock, Andrew (May 18, 2011). "Dr. Dobb's: Interview with Ken Thompson" (http://www.drdobbs.com/open-source/interview-with-ken-thompson/229502480). Retrieved February 7, 2014.
24. Pike, Rob (2012). "Less is exponentially more" (http://commandcenter.blogspot.mx/2012/06/less-is-exponentially-more.html).
25. Griesemer, Robert (2015). "The Evolution of Go" (https://talks.golang.org/2015/gophercon-goevolution.slide#4).
26. Griesemer, Robert; Pike, Rob; Thompson, Ken; Taylor, Ian; Cox, Russ; Kim, Jini; Langley, Adam. "Hey! Ho! Let's Go!" (https://opensource.googleblog.com/2009/11/hey-ho-lets-go.html). *Google Open Source*. Retrieved May 17, 2018.
27. Shankland, Stephen (March 30, 2012). "Google's Go language turns one, wins a spot at YouTube: The lower-level programming language has matured enough to sport the 1.0 version number. And it's being used for real work at Google" (https://www.cnet.com/news/googles-go-language-turns-one-wins-a-spot-at-youtube/). News. *CNet*. CBS Interactive Inc. Retrieved August 6, 2017. "Google has released version 1 of its Go programming language, an ambitious attempt to improve upon giants of the lower-level programming world such as C and C++."
28. "Release History - The Go Programming Language" (https://golang.org/doc/devel/release.html). *golang.org*.
29. "Go FAQ: Is Google using Go internally?" (https://golang.org/doc/faq#internal_usage). Retrieved March 9, 2013.

30. "Go fonts – The Go Blog" (https://blog.golang.org/go-fonts). Go. November 16, 2016. Retrieved March 12, 2019.
31. "Go Font TTFs" (https://github.com/golang/image/tree/master/font/gofont/ttfs). *GitHub*. Retrieved April 2, 2019.
32. "Go's New Brand" (https://blog.golang.org/go-brand). *The Go Blog*. Retrieved November 9, 2018.
33. "Go 2 Draft Designs" (https://go.googlesource.com/proposal/+/master/design/go2draft.md). Retrieved September 12, 2018.
34. "The Go Blog: Go 2 Draft Designs" (https://blog.golang.org/go2draft). August 28, 2018.
35. "Go 1 and the Future of Go Programs - The Go Programming Language" (https://golang.org/doc/go1compat). *golang.org*.
36. "Go 1.17 Release Notes - The Go Programming Language" (https://golang.org/doc/go1.17#introduction). *golang.org*.
37. "Release History - The Go Programming Language" (https://golang.org/doc/devel/release.html#policy). *golang.org*.
38. "Release History" (https://golang.org/doc/devel/release.html). *The Go Programming Language*. Retrieved August 24, 2018.
39. "Go 1.11 Release Notes – the Go Programming Language" (https://golang.org/doc/go1.11).
40. "Understanding Golang TLS mutual authentication DoS – CVE-2018-16875" (https://apisecurity.io/mutual-tls-authentication-vulnerability-in-go-cve-2018-16875/). December 19, 2018.
41. "Working with Errors in Go 1.13 - The Go Blog" (https://blog.golang.org/go1.13-errors). *blog.golang.org*. Retrieved March 11, 2021. "Go 1.13 introduces new features to the errors and fmt standard library packages to simplify working with errors that contain other errors. The most significant of these is a convention rather than a change: an error which contains another may implement an Unwrap method returning the underlying error. If e1.Unwrap() returns e2, then we say that e1 wraps e2, and that you can unwrap e1 to get e2."
42. "Go 1.14 Release Notes – the Go Programming Language" (https://golang.org/doc/go1.14).
43. "Go 1.15 is released - The Go Blog" (https://blog.golang.org/go1.15). *blog.golang.org*. Retrieved May 11, 2021. "Go 1.15 includes a new package, time/tzdata, that permits embedding the timezone database into a program. Importing this package (as import _ "time/tzdata") permits the program to find timezone information even if the timezone database is not available on the local system. You can also embed the timezone database by building with -tags timetzdata. Either approach increases the size of the program by about 800 KB"
44. "Go 1.15 Release Notes - The Go Programming Language" (https://golang.org/doc/go1.15#library). *golang.org*. Retrieved May 11, 2021.
45. "Go 1.16 Release Notes - The Go Programming Language" (https://golang.org/doc/go1.16). *golang.org*. Retrieved May 11, 2021.
46. "Go 1.17 Release Notes - The Go Programming Language" (https://golang.org/doc/go1.17). *golang.org*. Retrieved October 30, 2021.
47. Pike, Rob. "The Go Programming Language" (https://www.youtube.com/watch?v=rKnDgT73v8s). YouTube. Retrieved July 1, 2011.
48. Pike, Rob (November 10, 2009). *The Go Programming Language* (https://www.youtube.com/watch?v=rKnDgT73v8s#t=8m53) (flv) (Tech talk). Google. Event occurs at 8:53.
49. "Download and install packages and dependencies - go - The Go Programming Language" (https://golang.org/cmd/go/#hdr-Download_and_install_packages_and_dependencies). See godoc.org (http://godoc.org) for addresses and documentation of some packages.
50. "GoDoc" (http://godoc.org). *godoc.org*.
51. Pike, Rob. "The Changelog" (http://5by5.tv/changelog/100) (Podcast).
52. "Go Programming Language Specification, §Semicolons" (https://golang.org/ref/spec#Semicolons). *golang.org*.

53. "Effective Go, §Semicolons" (https://golang.org/doc/effective_go.html#semicolons). *golang.org*.
54. "The Go Programming Language Specification - The Go Programming Language" (https://golang.org/ref/spec#For_statements). *golang.org*.
55. Pike, Rob (October 23, 2013). "Strings, bytes, runes and characters in Go" (https://blog.golang.org/strings).
56. Doxsey, Caleb. "Structs and Interfaces — An Introduction to Programming in Go" (https://www.golang-book.com/books/intro/9). *www.golang-book.com*. Retrieved October 15, 2018.
57. Pike, Rob (September 26, 2013). "Arrays, slices (and strings): The mechanics of 'append' " (https://blog.golang.org/slices). *The Go Blog*. Retrieved March 7, 2015.
58. Gerrand, Andrew. "Go Slices: usage and internals" (https://blog.golang.org/go-slices-usage-and-internals).
59. The Go Authors. "Effective Go: Slices" (https://golang.org/doc/effective_go.html#slices).
60. The Go authors. "Selectors - The Go Programming Language Specification" (https://golang.org/ref/spec#Selectors).
61. The Go authors. "Calls - The Go Programming Language Specification" (https://golang.org/ref/spec#Calls).
62. "Go Programming Language Specification, §Package unsafe" (https://golang.org/ref/spec#Package_unsafe). *golang.org*.
63. "The Go Programming Language Specification" (https://golang.org/ref/spec#Assignability). *golang.org*.
64. "The Go Programming Language Specification" (https://golang.org/ref/spec#Constants). *golang.org*.
65. "The Go Programming Language Specification" (https://golang.org/ref/spec#Calls). *golang.org*.
66. Schmager, Frank; Cameron, Nicholas; Noble, James (2010). *GoHotDraw: evaluating the Go programming language with design patterns*. Evaluation and Usability of Programming Languages and Tools. ACM.
67. Summerfield, Mark (2012). *Programming in Go: Creating Applications for the 21st Century*. Addison-Wesley.
68. Balbaert, Ivo (2012). *The Way to Go: A Thorough Introduction to the Go Programming Language*. iUniverse.
69. "The Evolution of Go" (https://talks.golang.org/2015/gophercon-goevolution.slide#19). *talks.golang.org*. Retrieved March 13, 2016.
70. Diggins, Christopher (November 24, 2009). "Duck Typing and the Go Programming Language" (http://www.drdobbs.com/architecture-and-design/duck-typing-and-the-go-programming-langu/228701527). Dr. Dobb's. Retrieved March 10, 2016.
71. Ryer, Mat (December 1, 2015). "Duck typing in Go" (https://medium.com/@matryer/golang-advent-calendar-day-one-duck-typing-a513aaed544d#.ebm7j81xu). Retrieved March 10, 2016.
72. "Frequently Asked Questions (FAQ) - The Go Programming Language" (https://golang.org/doc/faq). *golang.org*.
73. "The Go Programming Language Specification" (https://golang.org/ref/spec#Type_assertions). *golang.org*.
74. "The Go Programming Language Specification" (https://golang.org/ref/spec#Type_switches). *golang.org*.
75. "reflect.ValueOf(i interface{}) converts an `interface{}` to a `reflect.Value` that can be further inspected" (https://golang.org/pkg/reflect/#ValueOf).
76. "map[string]interface{} in Go" (https://bitfieldconsulting.com/golang/map-string-interface). *bitfieldconsulting.com*.
77. "Go Data Structures: Interfaces" (http://research.swtch.com/interfaces). Retrieved November 15, 2012.

78. "The Go Programming Language Specification" (https://golang.org/ref/spec#Interface_types). *golang.org*.
79. "A Tutorial for the Go Programming Language" (https://golang.org/doc/go_tutorial.html). *The Go Programming Language*. Retrieved March 10, 2013. "In Go the rule about visibility of information is simple: if a name (of a top-level type, function, method, constant or variable, or of a structure field or method) is capitalized, users of the package may see it. Otherwise, the name and hence the thing being named is visible only inside the package in which it is declared."
80. "go - The Go Programming Language" (https://golang.org/cmd/go/#hdr-Download_and_install_packages_and_dependencies). *golang.org*.
81. "How to Write Go Code" (https://golang.org/doc/code.html). *golang.org*. "The packages from the standard library are given short import paths such as "fmt" and "net/http". For your own packages, you must choose a base path that is unlikely to collide with future additions to the standard library or other external libraries. If you keep your code in a source repository somewhere, then you should use the root of that source repository as your base path. For instance, if you have an Example account at example.com/user, that should be your base path"
82. "Go Packaging Proposal Process" (https://docs.google.com/document/d/18tNd8r5DV0yluCR7tPvkMTsWD_lYcRO7NhpNSDymRr8/edit?pref=2&pli=1&usp=embed_facebook). *Google Docs*.
83. Pike, Rob. "Concurrency is not Parallelism" (https://vimeo.com/49718712).
84. Chisnall, David (2012). *The Go Programming Language Phrasebook* (https://books.google.com/books?id=scyH562VXZUC). Addison-Wesley. ISBN 9780132919005.
85. "Effective Go" (https://golang.org/doc/effective_go.html#sharing). *golang.org*.
86. "The Go Memory Model" (https://golang.org/ref/mem). Retrieved April 10, 2017.
87. "Go Concurrency Patterns" (https://talks.golang.org/2012/concurrency.slide). *golang.org*.
88. Graham-Cumming, John. "Recycling Memory Buffers in Go" (https://blog.cloudflare.com/recycling-memory-buffers-in-go).
89. "tree.go" (https://golang.org/doc/play/tree.go).
90. Cheslack-Postava, Ewen. "Iterators in Go" (http://ewencp.org/blog/golang-iterators/).
91. Kernighan, Brian W. "A Descent Into Limbo" (http://www.vitanuova.com/inferno/papers/descent.html).
92. "The Go Memory Model" (https://golang.org/doc/go_mem.html). Retrieved January 5, 2011.
93. Tang, Peiyi (2010). *Multi-core parallel programming in Go* (http://www.ualr.edu/pxtang/papers/acc10.pdf) (PDF). Proc. First International Conference on Advanced Computing and Communications.
94. Nanz, Sebastian; West, Scott; Soares Da Silveira, Kaue. *Examining the expert gap in parallel programming* (http://se.inf.ethz.ch/people/west/expert-gap-europar-2013.pdf) (PDF). Euro-Par 2013. CiteSeerX 10.1.1.368.6137 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.368.6137).
95. Cox, Russ. "Off to the Races" (http://research.swtch.com/gorace).
96. Pike, Rob (October 25, 2012). "Go at Google: Language Design in the Service of Software Engineering" (https://talks.golang.org/2012/splash.article). Google, Inc. "There is one important caveat: Go is not purely memory safe in the presence of concurrency."
97. "Frequently Asked Questions (FAQ) - the Go Programming Language" (https://golang.org/doc/faq).
98. "A Story of a Fat Go Binary" (https://medium.com/@jondot/a-story-of-a-fat-go-binary-20edc6549b97). September 21, 2018.
99. "Go Programming Language Specification, §Interface types" (https://golang.org/ref/spec#Interface_types). *golang.org*.
100. "Go Programming Language Specification, §Type assertions" (https://golang.org/ref/spec#Type_assertion). *golang.org*.

101. *All Systems Are Go* (http://www.informit.com/articles/article.aspx?p=1623555). *informIT*. August 17, 2010. Retrieved June 21, 2018.
102. "E2E: Erik Meijer and Robert Griesemer – Going Go" (http://channel9.msdn.com/Blogs/Charles/Erik-Meijer-and-Robert-Griesemer-Go). *Channel 9*. Microsoft. May 7, 2012.
103. Pike, Rob. "Generating code" (https://blog.golang.org/generate).
104. "Type Parameters - Draft Design" (https://go.googlesource.com/proposal/+/refs/heads/master/design/go2draft-type-parameters.md). *go.googlesource.com*.
105. "Generics in Go" (https://bitfieldconsulting.com/golang/generics). *bitfieldconsulting.com*.
106. "Panic And Recover" (https://code.google.com/p/go-wiki/wiki/PanicAndRecover). Go wiki.
107. "Weekly Snapshot History" (https://golang.org/doc/devel/weekly.html#2010-03-30). *golang.org*.
108. "Proposal for an exception-like mechanism" (https://groups.google.com/group/golang-nuts/browse_thread/thread/1ce5cd050bb973e4). *golang-nuts*. March 25, 2010. Retrieved March 25, 2010.
109. "Effective Go" (https://golang.org/doc/effective_go.html#panic). *golang.org*.
110. "gofmt - The Go Programming Language" (https://golang.org/cmd/gofmt/). *golang.org*. Retrieved February 5, 2021.
111. "Effective Go" (https://golang.org/doc/effective_go.html). *golang.org*.
112. "Unused local variables" (https://yourbasic.org/golang/unused-local-variables/). *yourbasic.org*. Retrieved February 11, 2021.
113. "Unused package imports" (https://yourbasic.org/golang/unused-imports/). *yourbasic.org*. Retrieved February 11, 2021.
114. "Code Review Comments" (https://github.com/golang/go/wiki/CodeReviewComments). Retrieved July 3, 2018.
115. "Talks" (https://talks.golang.org/). Retrieved July 3, 2018.
116. "Errors Are Values" (https://blog.golang.org/errors-are-values). Retrieved July 3, 2018.
117. "fmt - The Go Programming Language" (https://golang.org/pkg/fmt/). *golang.org*. Retrieved April 8, 2019.
118. "testing - The Go Programming Language" (https://golang.org/pkg/testing/). *golang.org*. Retrieved December 27, 2020.
119. *avelino/awesome-go: A curated list of awesome Go frameworks, libraries and software* (https://github.com/avelino/awesome-go), retrieved January 10, 2018
120. "EdgeX Foundry Project" (https://github.com/edgexfoundry). *GitHub*. Retrieved February 6, 2021.
121. "ipfs/go-ipfs" (https://github.com/ipfs/go-ipfs). *GitHub*. Retrieved June 1, 2018.
122. "lightningnetwork/lnd" (https://github.com/lightningnetwork/lnd), *GitHub*, retrieved April 29, 2020
123. "NATS - Open Source Messaging System | Secure, Native Cloud Application Development" (https://nats.io).
124. "Test driven development in Go | Cacoo" (https://cacoo.com/blog/test-driven-development-in-go/). *Cacoo*. July 29, 2016. Retrieved June 1, 2018.
125. "Chango" (https://github.com/chango/). *GitHub*.
126. Heller, Martin (July 17, 2014). "Review: Cloud Foundry brings power and polish to PaaS" (https://www.javaworld.com/article/2455358/cloud-computing/review-cloud-foundry-brings-power-and-polish-to-paas.html). *JavaWorld*. Retrieved January 22, 2019.
127. Graham-Cumming, John. "Go at CloudFlare" (https://blog.cloudflare.com/go-at-cloudflare).
128. Graham-Cumming, John. "What we've been doing with Go" (https://blog.cloudflare.com/what-weve-been-doing-with-go).
129. "Go at CoreOS" (https://blog.gopheracademy.com/birthday-bash-2014/go-at-coreos/). November 25, 2014.

130. "Couchbase" (https://github.com/couchbase). *GitHub*.
131. Lee, Patrick (July 7, 2014). "Open Sourcing Our Go Libraries" (https://tech.dropbox.com/2014/07/open-sourcing-our-go-libraries/).
132. "Official Go implementation of the Ethereum protocol" (https://github.com/ethereum/go-ethereum). *GitHub*. ethereum. April 18, 2020.
133. "Why we use Ruby on Rails to build GitLab" (https://about.gitlab.com/blog/2018/10/29/why-we-use-rails-to-build-gitlab/). *GitLab*. Retrieved February 6, 2021. "Ruby was optimized for the developer, not for running it in production," says Sid. "For the things that get hit a lot and have to be very performant or that, for example, have to wait very long on a system IO, we rewrite those in Go ... We are still trying to make GitLab use less memory. So, we'll need to enable multithreading. When we developed GitLab that was not common in the Ruby on Rails ecosystem. Now it's more common, but because we now have so much code and so many dependencies, it's going to be a longer path for us to get there. That should help; it won't make it blazingly fast, but at least it will use less memory"
134. "dl.google.com: Powered by Go" (https://talks.golang.org/2013/oscon-dl.slide). *golang.org*.
135. Welsh, Matt. "Rewriting a Large Production System in Go" (http://matt-welsh.blogspot.com/2013/08/rewriting-large-production-system-in-go.html).
136. Symonds, David. "High Performance Apps on Google App Engine" (https://talks.golang.org/2013/highperf.slide).
137. "Mongo DB" (https://github.com/mongodb/mongo-tools#building-tools). *GitHub*. April 18, 2020.
138. "The Netflix Tech Blog: Application data caching using SSDs" (https://netflixtechblog.com/application-data-caching-using-ssds-5bf25df851ef). May 25, 2016.
139. "golang/go" (https://github.com/golang/go/wiki/GoUsers). *GitHub*. April 18, 2020.
140. Sacks, Steven. "Search & Advances" (https://web.archive.org/web/20150611115444/http://tech.plug.dj/2015/06/09/search-advances/). *plug.dj tech blog*. Archived from the original (https://tech.plug.dj/2015/06/09/search-advances/) on June 11, 2015. Retrieved June 10, 2015.
141. Jenkins, Tim (March 6, 2014). "How to Convince Your Company to Go With Golang" (https://sendgrid.com/blog/convince-company-go-golang/). *SendGrid's Email Deliverability Blog*.
142. Bourgon, Peter. "Go at SoundCloud" (https://web.archive.org/web/20131111191620/http://backstage.soundcloud.com/2012/07/go-at-soundcloud/). Archived from the original (https://backstage.soundcloud.com/2012/07/go-at-soundcloud/) on November 11, 2013.
143. "Go at Google I/O and Gopher SummerFest - The Go Blog" (https://blog.golang.org/io2014). *golang.org*.
144. TWSTRIKE (April 17, 2020). "CoyIM" (https://github.com/twstrike/coyim/). *ThoughtWorks STRIKE team*.
145. Hiltner, Rhys (July 5, 2016). "Go's march to low-latency GC" (https://blog.twitch.tv/gos-march-to-low-latency-gc-a6fa96f06eb7#.wykex6pkr).
146. "How We Built Uber Engineering's Highest Query per Second Service Using Go" (https://eng.uber.com/go-geofence/). *Uber Engineering Blog*. February 24, 2016. Retrieved March 2, 2016.
147. Simionato, Michele (November 15, 2009). "Interfaces vs Inheritance (or, watch out for Go!)" (http://www.artima.com/weblogs/viewpost.jsp?thread=274019). artima. Retrieved November 15, 2009.
148. Astels, Dave (November 9, 2009). "Ready, Set, Go!" (https://www.engineyard.com/blog/ready-set-go). engineyard. Retrieved November 9, 2009.
149. jt (January 11, 2010). "Google's Go Wins Programming Language Of The Year Award" (http://jaxenter.com/google-s-go-wins-programming-language-of-the-year-award-10069.html). jaxenter. Retrieved December 5, 2012.
150. "TIOBE Programming Community Index for June 2015" (http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html). TIOBE Software. June 2015. Retrieved July 5, 2015.
151. Eckel, Bruce (August 27, 2011). "Calling Go from Python via JSON-RPC" (http://www.artima.com/weblogs/viewpost.jsp?thread=333589). Retrieved August 29, 2011.

152. Hundt, Robert (2011). *Loop recognition in C++/Java/Go/Scala* (https://days2011.scala-lang.org/sites/days2011/files/ws3-1-Hundt.pdf) (PDF). Scala Days.
153. Metz, Cade (July 1, 2011). "Google Go strikes back with C++ bake-off" (https://www.theregister.co.uk/2011/07/01/go_v_cpluplus_redux/). *The Register*.
154. Brownlee, John (November 13, 2009). "Google didn't google "Go" before naming their programming language' " (https://web.archive.org/web/20151208143907/http://www.geek.com/news/google-didnt-google-go-before-naming-their-programming-language-977351/). Archived from the original (http://www.geek.com/news/google-didnt-google-go-before-naming-their-programming-language-977351/) on December 8, 2015. Retrieved May 26, 2016.
155. Claburn, Thomas (November 11, 2009). "Google 'Go' Name Brings Accusations Of Evil' " (http://www.informationweek.com/news/software/web_services/showArticle.jhtml?articleID=221601351). InformationWeek. Retrieved January 18, 2010.
156. "Issue 9 - go — I have already used the name for *MY* programming language" (https://github.com/golang/go/issues/9#issuecomment-66047478). *Github*. Google Inc. Retrieved October 12, 2010.
157. Yager, Will. "Why Go is not Good" (http://yager.io/programming/go.html). Retrieved November 4, 2018.
158. Elbre, Egon. "Summary of Go Generics discussions" (https://docs.google.com/document/d/1vrAy9gMpMoS3uaVphB32uVXX4pi-HnNjkMEgyAHX4N4/preview). Retrieved November 4, 2018.
159. Dobronszki, Janos. "Everyday Hassles in Go" (https://crufter.com/everyday-hassles-in-go). Retrieved November 4, 2018.
160. Fitzpatrick, Brad. "Go: 90% Perfect, 100% of the time" (https://talks.golang.org/2014/gocon-tokyo.slide#50). Retrieved January 28, 2016.
161. "Why are there braces but no semicolons? And why can't I put the opening brace on the next line?" (https://golang.org/doc/faq#semicolons). Retrieved March 26, 2020. "The advantages of a single, programmatically mandated format for all Go programs greatly outweigh any perceived disadvantages of the particular style."
162. "I want off Mr. Golang's Wild Ride" (https://fasterthanli.me/articles/i-want-off-mr-golangs-wild-ride). February 28, 2020. Retrieved November 17, 2020.
163. "proposal: os: Create/Open/OpenFile() set FILE_SHARE_DELETE on windows #32088" (https://github.com/golang/go/issues/32088). May 16, 2019. Retrieved November 17, 2020.
164. Tu, Tengfei (2019). "Understanding Real-World Concurrency Bugs in Go" (https://songlh.github.io/paper/go-study.pdf) (PDF). "For example, around 58% of blocking bugs are caused by message passing. In addition to the violation of Go's channel usage rules (e.g., waiting on a channel that no one sends data to or close), many concurrency bugs are caused by the mixed usage of message passing and other new semantics and new libraries in Go, which can easily be overlooked but hard to detect"

Further reading

- Donovan, Alan; Kernighan, Brian (October 2015). *The Go Programming Language* (https://www.informit.com/store/go-programming-language-9780134190440) (1st ed.). Addison-Wesley Professional. p. 400. ISBN 978-0-13-419044-0.

External links

- Official website (https://go.dev)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Go_(programming_language)&oldid=1058836981"