Go (programming language)

For the agent-based language released in 2003, see Go! (programming language).
"Google Go" redirects here. For the computer program by Google to play the board game Go, see AlphaGo.

Go (often referred to as **golang**) is a free and open source[12] programming language created at Google[13] in 2007 by Robert Griesemer, Rob Pike, and Ken Thompson.[10] It is a compiled, statically typed language in the tradition of Algol and C, with garbage collection, limited structural typing,[3] memory safety features and CSP-style concurrent programming features added.[14]

The language was announced in November 2009; it is used in some of Google's production systems,[15] as well as by other firms. Two major implementations exist: Google's Go compiler, "gc", is developed as open source software and targets various platforms including Linux, OS X, Windows, various BSD and Unix versions, and since 2015 also mobile devices, including smartphones.[16] A second compiler, gccgo, is a GCC frontend.[17][18] The "gc" toolchain is self-hosting since version 1.5.[19]

1 History

Go originated as an experiment by Google engineers Robert Griesemer, Rob Pike, and Ken Thompson to design a new programming language that would resolve common criticisms of other languages while maintaining their positive characteristics.[20] The new language was to:

- be statically typed, scalable to large systems (as Java and C++);
- be productive and readable, without too many mandatory keywords and repetition[21] ("light on the page" like dynamic languages);
- not require tooling, but support it well,
- support networking and multiprocessing.

In later interviews, all three of the language designers cited their shared dislike of C++'s complexity as a primary motivation for designing a new language.[22][23][24]

Go 1.0 was released in March 2012.[25]

Go 1.7 added "one tiny language change"[26] and one port to macOS 10.12 Sierra plus some experimental ports, e.g. for Linux on z Systems (linux/s390x). Some library changes apply, and e.g. Unicode 9.0 is now supported.

2 Language design

Go is recognizably in the tradition of C, but makes many changes to improve conciseness, simplicity, and safety. The following is a brief overview of the features which define Go:

- A syntax and environment adopting patterns more common in dynamic languages:[27]
  - Optional concise variable declaration and initialization through type inference (x := 0 not var x int = 0;).
  - Fast compilation times.[28]
  - Remote package management (go get)[29] and online package documentation.[30]
- 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,[31] in part by omitting features common to similar languages.

2.1 Criticism

Go critics assert that:

- lack of compile-time generics leads to code duplication, metaprogramming cannot be statically checked[32][33] and standard library cannot offer generic algorithms[34]

- lack of language extensibility (through, for instance, operator overloading) makes certain tasks more verbose[32][35]
- the type system's lack of Hindley-Milner typing inhibiting safety and/or expressiveness[36][37]
- the pauses and overhead of garbage collection (*GC*) limit Go's use in systems programming compared to languages with manual memory management.[32][36]

The language designers argue that these trade-offs are important to Go's success,[38] and explain some particular decisions at length,[39] though they do express openness to adding some form of generic programming in the future, and to pragmatic improvements in areas like standardizing ways to apply code generation.[40] Regarding GC, Go defenders point to pause-time reduction in later versions[41] (e.g. Go 1.6), while acknowledging their GC algorithm is not hard real-time.

2.2 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 := "some words", without specifying the types of variables. This contrasts with C's int i = 3; and const char *s = "some words";. Semicolons still terminate statements, but are implicit when they would occur at the end of a line. Functions may return multiple values, and returning a result, err pair is the conventional way a function indicates an error to its caller in Go.[lower-alpha 1] 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.

2.3 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 and decoding.[42] Record types can be defined with the struct keyword.

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.[43][44][45]

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.[46] There is no pointer arithmetic, except via the special unsafe.Pointer type in the standard library.

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.[47] For example, the type keyword can be used to define a type for IPv4 addresses, which are 32-bit unsigned integers:

```
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.[48]

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 converting an address to a human-readable representation, viz.,

```
// Is this the zero broadcast address 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.[49]

2.3.1 Interface system

Go provides two features that replace class inheritance.

The first is *embedding*, which can be viewed as an auto-

mated form of composition[50] or delegation.[51]:255

The second are its *interfaces*, which provides runtime polymorphism.[52]:266 Interfaces provide a limited form of structural typing in the otherwise nominal type system of Go. Any type that implements all methods of an interface conforms to that interface. Go interfaces were designed after protocols from the Smalltalk programming language.[53] Multiple sources use the term duck typing when describing Go interface.[54][55] 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 to use the term structural typing.

An interface specifies a set of types by listing required methods and their types, and is satisfied by any type that has the required methods. Implementing types do not need to specify their implementing of interfaces, so if Shape, Square and Circle are defined as:

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 Square and Circle are implicitly a Shape and can be assigned to a Shape-typed variable.[52]: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.[52]: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.[52]: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*,[56] which checks against a single potential type, and the *type switch*,[57] which checks against multiple types.

The *empty interface* interface{} is an important corner case because it can refer to an item of *any* concrete type. It is similar to the Object class in Java or C#, but with the difference that the empty interface is satisfied by any type, including built-in types like int (while in Java and C#, an Object variable can only hold instances of reference type).[52]: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.[58] 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.

Interface values are implemented using pointer to data and a second pointer to run-time type information.[59] Like some other types implemented using pointers in Go, interface values are nil if uninitialized.[60]

2.4 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.[61] The go get command can retrieve packages stored in a remote repository such as GitHub.,[62] and developers are encouraged to develop packages inside a base path corresponding to a source repository (such as github.com/user_name/package_name) to reduce the likelihood of name collision with future additions to the standard library or other external libraries.[63]

Proposals exist to introduce a proper package management solution for Go similar to Rust's cargo system or Node's npm system.[64]

2.5 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.[65]

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.[66]:10

While a standard library package featuring most of the classical concurrency control structures (mutex locks, etc.) is available,[66]:151–152 idiomatic concurrent programs instead prefer *channels*, which provide send messages between goroutines.[67] Optional buffers store messages in FIFO order[51]: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 expres-

sion 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.

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.[66]: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.[68] Channels have also found uses further from the usual notion of interprocess communication, like serving as a concurrency-safe list of recycled buffers,[69] implementing coroutines (which helped inspire the name *goroutine*),[70] and implementing iterators.[71]

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),[72] Go does not provide any built-in notion of safe or verifiable concurrency.[73] 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.

2.5.1 Suitability for parallel programming

Although Go's concurrency features are not aimed primarily at parallel processing,[65] they can be used to program shared memory multi-processor machines. Various studies have been done into the effectiveness of this approach.[74] 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.[75]

2.5.2 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.[73] 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.[76][77]

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.[66]:155

2.6 Omissions

Go deliberately omits certain features common in other languages, including (implementation) inheritance, generic programming, assertions, pointer arithmetic, and implicit type conversions.

Of these language features, the Go authors express an openness to generic programming, 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[lower-alpha 2] 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 not overuse it to the extent that programmers in other languages are reputed to overuse inheritance.[50]

Regarding generic programming, some built-in functions *are* in fact type-generic, but these are treated as special cases; Rob Pike calls this a weakness of the language that may at some point be changed.[43] The Google team that designs the language built at least one compiler for an experimental Go dialect with generics, but did not release it.[78]

After initially omitting exceptions, the exception-like panic/recover mechanism was eventually added to the language, 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).[79][80][81][82]

3 Conventions and code style

The Go authors put substantial effort into molding the style and design of Go programs:

- Indentation, spacing, and other surface-level details of code are automatically standardized by the gofmt tool. golint does additional style checks automatically.
- Tools and libraries distributed with Go suggest standard approaches to things like API documentation (godoc[83]), 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 or imports, and implicit type conversions.
- The *omission* of certain features (for example, functional-programming shortcuts like map and C++-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, and later also collected code review comments, talks, official blog posts to teach Go style and coding philosophy.

4 Language tools

Go includes the same sort of debugging, testing, and code-vetting tools as many language distributions. The Go distribution includes, among other tools,

- 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 get, 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 (to, for example, track garbage collection pauses), and a race condition tester.

There is an ecosystem of third-party tools that add to the standard distribution, such as gocode, which enables code autocompletion in many text editors, goimports (by a Go team member), which automatically adds/removes package imports as needed, errcheck, which detects code that might unintentionally ignore errors, and more. Plugins exist to add language support in widely used text editors, and at least one IDE, LiteIDE, is branded as "a simple, open source, cross-platform Go IDE."[84]

5 Examples

5.1 Hello world

Here is a Hello world program in Go:

```
package main import "fmt" func main() { fmt.Println("Hello, World") }
```

5.2 Concurrency example

The following simple program demonstrates Go's concurrency features to implement an asynchronous program. It launches two "goroutines" (lightweight threads): 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 Chisnall).[66]: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 <- true } 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.") } }
```

6 Projects using Go

Some notable open-source applications in Go include:

- Docker, a set of tools for deploying Linux containers
- Doozer, a lock service by managed hosting provider Heroku[14]

- Ethereum, a shared world computing platform.
- Juju, a service orchestration tool by Canonical, packagers of Ubuntu Linux
- Packer, a tool for creating identical machine images for multiple platforms from a single source configuration
- Snappy, a package manager developed by Canonical for Ubuntu.
- Syncthing, an open-source file synchronization client/server application

Some notable open-source frameworks using Go:

- Beego, high-performance web framework in Go, used for web apps and backend services.
- Martini, package for web applications/services.
- Gorilla, a web toolkit for Go.

Other notable companies and sites using Go (generally together with other languages, not exclusively) include:[85][86]

- AeroFS, private cloud filesync appliance provider which migrated some microservices from Java to Go with major memory footprint improvements[87]
- Chango, a programmatic advertising company uses Go in its real-time bidding systems.[88]
- Cloud Foundry, a platform as a service
- CloudFlare, for their delta-coding proxy Railgun, their distributed DNS service, as well as tools for cryptography, logging, stream processing, and accessing SPDY sites.[89][90]
- CoreOS, a Linux-based operating system that utilizes Docker containers.[91]
- Couchbase, Query and Indexing services within the Couchbase Server[92]
- Dropbox, migrated some of their critical components from Python to Go[93]
- Google, for many projects, notably including download server dl.google.com[94][95][96]
- MercadoLibre, for several public APIs.
- MongoDB, tools for administering MongoDB instances[97]
- Netflix, for two portions of their server architecture[98]
- Novartis, for an internal inventory system
- Plug.dj, an interactive online social music streaming website.[99]
- Replicated, Docker based PaaS for creating enterprise, installable software.[100]
- SendGrid, a Boulder, Colorado-based transactional email delivery and management service.[101]
- SoundCloud, for "dozens of systems"[102]
- Splice, for the entire backend (API and parsers) of their online music collaboration platform.[103]
- ThoughtWorks, some tools and applications around continuous delivery and instant messages (CoyIM).[104]
- Twitch.tv, for their IRC-based chat system (migrated from Python).[105]
- Uber, for handling high volumes of geofence-based queries.[106]
- Zerodha, for realtime peering and streaming of market data

7 Reception

Go's initial release led to much discussion.

The interface system, and the deliberate omission of inheritance, were praised by Michele Simionato, who likened these language characteristics to those of Standard ML, calling it "a shame that no popular language has followed [this] particular route in the design space".[107]

Dave Astels at Engine Yard wrote:[108]

> 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.

Ars Technica interviewed Rob Pike, one of the authors of Go, and asked why a new language was needed. He replied that:[109]

> It wasn't enough to just add features to existing programming languages, because sometimes you can get more in the long run by taking things away. They wanted to start from scratch and rethink everything. ... [But they did not want] to deviate too much from what developers already knew because they wanted to avoid alienating Go's target audience.

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,[110] 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.[111] But as of September 2016, its ranking had surged to 19th, indicating significant growth in popularity and adoption.[112]

Regarding Go, Bruce Eckel has stated:[113]

> 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 that:

> 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[114]

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 did optimize 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 "optimized" code in the paper.[115]

8 Naming dispute

On 10 November 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.[116] 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[117] even saying the issue contradicts Google's motto of: Don't be evil.[118] The issue was closed by a Google developer on 12 October 2010 with the custom status "Unfortunate" and with 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."

9 See also

- Comparison of programming languages
- Dart, another Google programming language
- UFCS, a way of having 'open methods' in other languages

10 Notes

[1] 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.

[2] Questions "How do I get dynamic dispatch of methods?" and "Why is there no type inheritance?" in the language FAQ.[10]

11 References

[1] "FAQ — The Go Programming Language". Golang.org. Retrieved 2013-06-25.

[2] "Release History - The Go Programming Language". *golang.org*. Retrieved 19 October 2016.

[3] "Why doesn't Go have "implements" declarations?". *golang.org*. Retrieved 1 October 2015.

[4] Pike, Rob (2014-12-22). "Rob Pike on Twitter". Retrieved 2016-03-13. Go has structural typing, not duck typing. Full interface satisfaction is checked and required.

[5] "lang/go: go-1.4 – Go programming language". *OpenBSD ports*. 2014-12-23. Retrieved 2015-01-19.

[6] "Go Porting Efforts". *Go Language Resources*. cat-v. 12 January 2010. Retrieved 18 January 2010.

[7] "Text file LICENSE". *The Go Programming Language*. Google. Retrieved 5 October 2012.

[8] "Additional IP Rights Grant". *The Go Programming Language*. Google. Retrieved 5 October 2012.

[9] Pike, Rob (2014-04-24). "Hello Gophers". Retrieved 2016-03-11.

[10] "Language Design FAQ". *golang.org*. 16 January 2010. Retrieved 27 February 2010.

[11] "The Evolution of Go". Retrieved 2015-09-26.

[12] https://golang.org/LICENSE

[13] Kincaid, Jason (10 November 2009). "Google's Go: A New Programming Language That's Python Meets C++". *TechCrunch*. Retrieved 18 January 2010.

[14] Metz, Cade (5 May 2011). "Google Go boldly goes where no code has gone before". *The Register*.

[15] "Go FAQ: Is Google using Go internally?". Retrieved 9 March 2013.

[16] "Google's In-House Programming Language Now Runs on Phones". *wired.com*. 19 August 2015.

[17] "FAQ: Implementation". *golang.org*. 16 January 2010. Retrieved 18 January 2010.

[18] "Installing GCC: Configuration". Retrieved 3 December 2011. Ada, Go and Objective-C++ are not default languages

[19] "Go 1.5 Release Notes". Retrieved 28 January 2016. The compiler and runtime are now implemented in Go and assembler, without C.

[20] Pike, Rob (28 April 2010). "Another Go at Language Design". *Stanford EE Computer Systems Colloquium*. Stanford University. Video available.

[21] "Frequently Asked Questions (FAQ) - The Go Programming Language". *golang.org*. Retrieved 2016-02-26.

[22] Andrew Binstock (18 May 2011). "Dr. Dobb's: Interview with Ken Thompson". Retrieved 7 February 2014.

[23] Pike, Rob (2012). "Less is exponentially more".

[24] Robert Griesemer (2015). "The Evolution of Go".

[25] "Release History".

[26] https://golang.org/doc/go1.7

[27] Pike, Rob. "The Go Programming Language". YouTube. Retrieved 1 Jul 2011.

[28] Rob Pike (10 November 2009). *The Go Programming Language* (flv) (Tech talk). Google. Event occurs at 8:53.

[29] Download and install packages and dependencies - go - The Go Programming Language; see godoc.org for addresses and documentation of some packages

[30] "GoDoc". *godoc.org*.

[31] Rob Pike, on The Changelog podcast

[32] Will Yager, Why Go is not Good

[33] Egon Elbre, Summary of Go Generics discussions

[34] Fitzpatrick, Brad. "Go: 90% Perfect, 100% of the time". Retrieved 28 January 2016.

[35] Danny Gratzer, Leaving Go

[36] Jared Forsyth, Rust vs. Go

[37] Janos Dobronszki, Everyday Hassles in Go

[38] Rob Pike, Less is exponentially more

[39] The Go Authors, Frequently Asked Questions (FAQ)

[40] Rob Pike, Generating code

[41] Richard Hudson, Go 1.4+ Garbage Collection (GC) Plan and Roadmap

[42] Rob Pike, Strings, bytes, runes and characters in Go, 23 October 2013

[43] Pike, Rob (26 September 2013). "Arrays, slices (and strings): The mechanics of 'append'". *The Go Blog*. Retrieved 7 March 2015.

[44] Andrew Gerrand, Go Slices: usage and internals

[45] The Go Authors, Effective Go: Slices

[46] The Go authors Selectors - The Go Programming Language Specification and Calls - The Go Programming Language Specification

[47] "The Go Programming Language Specification". *golang.org*.

[48] "The Go Programming Language Specification". *golang.org*.

[49] "The Go Programming Language Specification". *golang.org*.

[50] 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.

[51] Summerfield, Mark (2012). *Programming in Go: Creating Applications for the 21st Century*. Addison-Wesley.

[52] Balbaert, Ivo (2012). *The Way to Go: A Thorough Introduction to the Go Programming Language*. iUniverse.

[53] "The Evolution of Go". *talks.golang.org*. Retrieved 2016-03-13.

[54] Diggins, Christopher (2009-11-24). "Duck Typing and the Go Programming Language". Dr. Dobb's. Retrieved 2016-03-10.

[55] Ryer, Mat (2015-12-01). "Duck typing in Go". Retrieved 2016-03-10.

[56] "The Go Programming Language Specification". *golang.org*.

[57] "The Go Programming Language Specification". *golang.org*.

[58] reflect.ValueOf(i interface{}) converts an interface{} to a reflect.Value that can be further inspected

[59] "Go Data Structures: Interfaces". Retrieved 15 November 2012.

[60] "The Go Programming Language Specification". *golang.org*.

[61] "A Tutorial for the Go Programming Language". *The Go Programming Language*. Google. Retrieved 10 March 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.

[62] "go - The Go Programming Language". *golang.org*.

[63] "How to Write Go Code". *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 a GitHub account at github.com/user, that should be your base path

[64] "Go Packaging Proposal Process".

[65] Rob Pike, Concurrency is not Parallelism

[66] Chisnall, David (2012). *The Go Programming Language Phrasebook*. Addison-Wesley.

[67] "Effective Go". *golang.org*.

[68] "Go Concurrency Patterns". *golang.org*.

[69] John Graham-Cumming, Recycling Memory Buffers in Go

[70] tree.go

[71] Ewen Cheslack-Postava, Iterators in Go

[72] Brian W. Kernighan, A Descent Into Limbo

[73] "The Go Memory Model". Google. Retrieved 5 January 2011.

[74] Tang, Peiyi (2010). *Multi-core parallel programming in Go* (PDF). Proc. First International Conference on Advanced Computing and Communications.

[75] Nanz, Sebastian; West, Scott; Soares Da Silveira, Kaue. *Examining the expert gap in parallel programming* (PDF). Euro-Par 2013. CiteSeerX 10.1.1.368.6137.

[76] Russ Cox, Off to the Races

[77] Rob Pike (October 25, 2012). "Go at Google: Language Design in the Service of Software Engineering". Google, Inc. "There is one important caveat: Go is not purely memory safe in the presence of concurrency."

[78] "E2E: Erik Meijer and Robert Griesemer – Going Go". *Channel 9*. Microsoft. 7 May 2012.

[79] Panic And Recover, Go wiki

[80] "Weekly Snapshot History". *golang.org*.

[81] "Proposal for an exception-like mechanism". *golang-nuts*. 25 March 2010. Retrieved 25 March 2010.

[82] "Effective Go". *golang.org*.

[83] "Effective Go". *golang.org*.

[84] LiteIDE,

[85] Erik Unger, The Case For Go

[86] Andrew Gerrand, Four years of Go, The Go Blog

[87] Hugues Bruant. "AeroFS - A little golang way". *AeroFS*.

[88] "Chango". *GitHub*.

[89] John Graham-Cumming, Go at CloudFlare

[90] John Graham-Cumming, What we've been doing with Go

[91] "Go at CoreOS".

[92] "Couchbase". *GitHub*.

[93] Patrick Lee, Open Sourcing Our Go Libraries, 7 July 2014.

[94] "dl.google.com: Powered by Go". *golang.org*.

[95] Matt Welsh, Rewriting a Large Production System in Go

[96] David Symonds, High Performance Apps on Google App Engine

[97] "Mongo DB". *GitHub*.

[98] "The Netflix Tech Blog: Application data caching using SSDs".

[99] Steven Sacks. "Search & Advances". *plug.dj tech blog*.

[100] "ReplicatedHQ". *GitHub*.

[101] Tim Jenkins. "How to Convince Your Company to Go With Golang". *SendGrid's Email Deliverability Blog*.

[102] Peter Bourgon, Go at SoundCloud

[103] "Go at Google I/O and Gopher SummerFest - The Go Blog". *golang.org*.

[104] TWSTRIKE. "CoyIM". *ThoughtWorks STRIKE team*.

[105] Rhys Hiltner, Go's march to low-latency GC, 5 July 2016.

[106] "How We Built Uber Engineering's Highest Query per Second Service Using Go". *Uber Engineering Blog*. Retrieved 2016-03-02.

[107] Simionato, Michele (15 November 2009). "Interfaces vs Inheritance (or, watch out for Go!)". artima. Retrieved 15 November 2009.

[108] Astels, Dave (9 November 2009). "Ready, Set, Go!". engineyard. Retrieved 9 November 2009.

[109] Paul, Ryan (10 November 2009). "Go: new open source programming language from Google". Ars Technica. Retrieved 13 November 2009.

[110] jt. "Google's Go Wins Programming Language Of The Year Award". jaxenter. Retrieved 5 December 2012.

[111] "TIOBE Programming Community Index for June 2015". TIOBE Software. June 2015. Retrieved 5 July 2015.

[112] "TIOBE Programming Community Index for September 2016". TIOBE Software. September 2016. Retrieved 17 September 2016.

[113] Bruce Eckel (27 August 2011). "Calling Go from Python via JSON-RPC". Retrieved 29 August 2011.

[114] Hundt, Robert (2011). *Loop recognition in C++/Java/Go/Scala* (PDF). Scala Days.

[115] Metz, Cade (1 July 2011). "Google Go strikes back with C++ bake-off". *The Register*.

[116] Brownlee, John (13 November 2009). "Google didn't google "Go" before naming their programming language'".

[117] Claburn, Thomas (11 November 2009). "Google 'Go' Name Brings Accusations Of Evil'". InformationWeek. Retrieved 18 January 2010.

[118] "Issue 9 - go — I have already used the name for *MY* programming language". *Google Code*. Google Inc. Retrieved 12 October 2010.

12 External links

- Official website
- A Tour of Go (official)
- Go Programming Language Resources (unofficial)

12.1 Community and conferences

- Gopher Academy, Gopher Academy is a group of developers working to educate and promote the Go community.
- Golangprojects.com, lists programming jobs and projects where companies are looking for people that know Go
- GopherCon The first Go conference. Denver, Colorado, USA April
- Gopher Gala The first Go hackathon.
- GopherConIndia The first Go conference in India. Bangalore Feb.
- GolangUK The first Go conference in UK. London
- dotGo European conference. Paris, France
- GolangShow Go audio podcast GolangShow
- Golang Argentina Gophers community blog in Argentina.

13 Text and image sources, contributors, and licenses

13.1 Text

- **Go (programming language)** *Source:* https://en.wikipedia.org/wiki/Go_(programming_language)?oldid=751516309 *Contributors:* Shii, Yves Junqueira, Twilsonb, Bdesham, Nealmcb, Chris~enwiki, TakuyaMurata, Julesd, Glenn, Rl, Andrewman327, Sanxiyn, Carol Fenijn, Samsara, Wdscxsj, Northgrove, PBS, Sverdrup, Filemon, Philwiki, Dissident, Dratman, Gracefool, Matt Crypto, Pgan002, Am088, Perey, Hydrox, Slipstream, Bender235, Syp, Kundor, Guy Harris, Prodicus, PatrickFisher, Stefan.karpinski, Tuggler, Tsuba~enwiki, Voxadam, Mindmatrix, Lost.goblin, Bkkbrad, Choas~enwiki, Waldir, BD2412, Qwertyus, Tlroche, XP1, Svaksha, Aidoor, Fragglet, Bgwhite, Peterl, Banaticus, Wavelength, Groogle, Petter Strandmark, ZacBowling, Jknacnud, Porttikivi, Amalthea, SmackBot, Hydrology, McGeddon, Brianski, Ohnoitsjamie, Hmains, JorgePeixoto, Ian13, Thumperward, Jeysaba, Letdorf, Nbarth, Sct72, Juancnuno, BarryNorton, Finx, Cybercobra, MureninC, Jonovision, Martijn Hoekstra, Elimisteve, DMacks, Daniel.Cardenas, Curly Turkey, ArglebargleIV, Twotwotwo, Soumya92, Michael miceli, Fig wright, Nagle, Loadmaster, RKT, Chickencha, Dl2000, Iridescent, Amniarix, Doceddi, Raysonho, MarsRover, Pgr94, ChristTrekker, Blaisorblade, Odie5533, JamesBrownJr, Hervegirod, Steve.ruckdashel, Rquesada, Stybn, Swehack, Widefox, Prolog, Hexene, Josephmarty, Slartidan, Magioladitis, RogierBrussee, DAGwyn, Vanished user ty12kl89jq10, Fshahriar, Abednigo, JaGa, Keith D, R'n'B, Garkbit, Sigmundur, Mdemare~enwiki, Deor, Mkcmkc, TXiKiBoT, Nxavar, Robennals, Tuxcantfly, PieterDeBruijn, Adamstac, Biasoli, Darxus, IndianGuru, Alexandre Bouthors, OsamaK, JonnyJD, SieBot, Meltonkt, Jerryobject, Flyer22 Reborn, Svick, Rahulrulez, Pianoman320, Gerardohc, WikiLaurent, Dom96, Stokito, Eposse, Mild Bill Hiccup, JJuran, Pdone, Pointillist, Alexbot, Glyn normington, IMneme, Davcamer, SF007, XLinkBot, Dsimic, Addbot, TutterMouse, Btx40, AndersBot, Zorrobot, Jarble, Fryedpeach, Luckas-bot, Yobot, Vanished user rt41as76lk, CSimons, Masharabinovich, AnomieBOT, Zephyrtronium, 1exec1, Wickorama, Citation bot, Charlieegan3, ArthurBot, Sidsel Sørendatter, Xqbot, Happyrabbit, RoodyAlien, Nasnema, Daviddengcn, Karlos77, ProfCoder, Xan2, MuffledThud, Gnuish, Aclassifier, FrescoBot, LucienBOT, Mu Mind, D'ohBot, Egmetcalfe, Diwas, Busukxuan, Jonesey95, RedBot, Gzhao, MondalorBot, Serols, Chulki Lee, Txt.file, TobeBot, Dchestnykh, ClaudeX, Skulldyvan, Lotje, Blue Em, Espadrine, JnRouvignac, ErikvanB, Isaac B Wagner, Genhuan, Melnakeeb, Wrackey, Ahmed Fatoum, .p2502, JonathanMayUK, Bounce1337, Lopifalko, Lt. John Harper, EmausBot, Goplexian, Wikipelli, Aeiuthhiet, Alfredo ougaowen, Veikk0.ma, Soni master, H3llBot, Wei2912, Mister Mormon, Aurelien.desbrieres, Drewlesueur, Cgt, ClueBot NG, Derick1259, Strcat, Kasirbot, Helpful Pixie Bot, மதனாஹரன், BG19bot, Grantmiller, ElphiBot, Kendall-K1, Compfreak7, Atomician, Exercisephys, Boshomi, Chmarkine, SnowdogU77, Sinysee, Fuse809, BattyBot, Pratyya Ghosh, Cpatch, Abledsoe78, Ender409, AlecTaylor, Rezonansowy, Rthangam77, Codename Lisa, Mogism, Bla5er89, Pintoch, Pokajanje, Mark viking, TvojaStara, Jcvernaleo, Tgrosinger, Mmautner, Myconix, Georgij Michaliutin, Matttproud, Comp.arch, Huihermit, Kintamanimatt, ShuklaSannidhya, Notrium, Stephenalexbrowne, Linuxjava, There is a T101 in your kitchen, Wikingr, Patrios, Hagis uk, U2fanboi, OMPIRE, Akwillis, Bwegs14, Ayberkt, Frontallappen, Nelsonkam, SarahTehCat, Nolk7, Rincevm, Aleksandar.d75, Sizeofint, Csousapt, Lerenn, Roblooby, Manojee87, Manthantri, Aznashwan, LexS007, Daniellacapriato, Hemanth7787, Norvoid, Daniele.baroncelli, Rajender999, Transfat0g, 1337Space, Jackie.his, Notnoyyyyy, Piotr Zaborski, BtySgtMajor, Fmadd, Random earthling, Expatriaticus and Anonymous: 320

13.2 Images

- **File:Commons-logo.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/4/4a/Commons-logo.svg *License:* CC-BY-SA-3.0 *Contributors:* ? *Original artist:* ?
- **File:Folder_Hexagonal_Icon.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/4/48/Folder_Hexagonal_Icon.svg *License:* Cc-by-sa-3.0 *Contributors:* ? *Original artist:* ?
- **File:Free_and_open-source_software_logo_(2009).svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/3/31/Free_and_open-source_software_logo_%282009%29.svg *License:* Public domain *Contributors:* FOSS Logo.svg *Original artist:* Free Software Portal Logo.svg (FOSS Logo.svg): ViperSnake151
- **File:Lock-green.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/6/65/Lock-green.svg *License:* CC0 *Contributors:* en:File:Free-to-read_lock_75.svg *Original artist:* User:Trappist the monk
- **File:Portal-puzzle.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/f/fd/Portal-puzzle.svg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Question_book-new.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/9/99/Question_book-new.svg *License:* Cc-by-sa-3.0 *Contributors:*

  Created from scratch in Adobe Illustrator. Based on Image:Question book.png created by User:Equazcion *Original artist:* Tkgd2007
- **File:Symbol_list_class.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/d/db/Symbol_list_class.svg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Symbol_neutral_vote.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/8/89/Symbol_neutral_vote.svg *License:* Public domain *Contributors:* ? *Original artist:* ?

13.3 Content license

- Creative Commons Attribution-Share Alike 3.0