

Go

A Programming Language

By Google

“golang”

History

- Go was designed at Google in 2007
- improve programming productivity in an era of multicore, networked machines and extremely large codebases.
- The designers were primarily motivated by their shared [dislike of C++](#)
- publicly announced in November 2009, and version 1.0 was released in March 2012
- Current version GO 1.13

Gopher image logo



Design

- Optional concise variable declaration and initialization through type inference (
 - `x := 0` not `int x = 0;` or `var x = 0;`).
- Fast compilation times.
- Remote package management (`go get`) and online package documentation.

Syntax changes from C

- 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.
- Semicolons are optional, only needed if you have more than one statement on a line otherwise EOL acts as semicolon.
- Functions may return multiple values
- As an alternative to C's three-statement for loop, Go's range expressions allow concise iteration over arrays, slices, strings, maps, and channels

Types in GO

- Go has a number of built-in types, including numeric ones (byte, int64, float32, etc.)
- 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
- There is no pointer arithmetic, except via the special **unsafe.Pointer** type in the standard library.
- 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.

Other facts.....

- Omissions: (implementation) inheritance, **generic programming**, assertions, pointer arithmetic, **implicit type conversions**, untagged unions, and tagged unions.
 - Assertions: Assert that this is always true at this point of execution.

```
x = 1;  
assert x > 0;  
x++;  
assert x > 1;
```
- Concurrency : There are no restrictions on how goroutines access shared data, making race conditions possible.

- exception-like panic/recover mechanism was eventually added

Hello World

```
package main
```

```
import "fmt"
```

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

Concurrency – Go is known for it's threads

The following simple program (2 slides) 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 David Chisnall book)

```
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.")  
    }  
}
```