



# Golang Programming

Introduction to Go

Go is an open source programming language that makes it easy to build simple, reliable, and efficient software.

Go web site at [golang.org](https://golang.org)

# Course Schedule

- [Jan 16, 2021] Introduction to Go
- [Jan 17, 2021] Program structure, data types, operators, control-flow statements, functions – *Homework 1 (algorithmic problem)*
- [Jan 23, 2021] Composite types, functions, error handling
- [Jan 24, 2021] Methods – *Homework 2 (Github JSON client) (3 h)*
- [Jan 30, 2021] Interfaces
- [Jan 31, 2021] Goroutines and Channels – *Homework 3 (building a simple web application)*
- [Feb 6, 2021] Concurrency with Shared Variables
- [Feb 7, 2021] Working with SQL and NoSQL databases – *Homework 4 (concurrent Web Scrapping)*

# Course Schedule

- [\[Feb 13, 2021\]](#) Building web services with REST
- [\[Feb 14, 2021\]](#) Building web services with gRPC
- [\[Feb 20, 2021\]](#) Building web services with GraphQL
- [\[Feb 21, 2021\]](#) Modules and dependency management using go mod – *Projects Mentoring*
- [\[Feb 27, 2021\]](#) Code generation. Summary
- [\[Mar 13, 2021\]](#) Projects demonstration
- [\[Mar 14, 2021\]](#) Receiving certificates

# Where to Find The Code and Materials?

<https://github.com/iproduct/coursego>

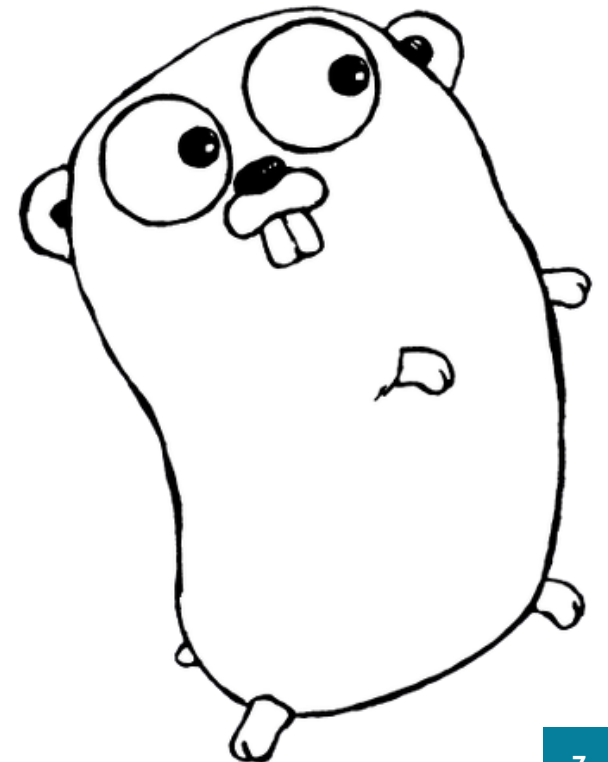
# Why Go?

History, main features, advantages



# Origins of GO

- Go was conceived in September 2007 by Robert Griesemer, Rob Pike, and Ken Thompson, at Google.
- It was publically announced in November 2009, and version 1.0 was released in March 2012.
- Go is widely used in production at Google and in many other organizations and open-source projects.





# Aims of Go

- The aim of Go language, was to fill the same niche today that C fit into in the '80s.
- According to Moore's law, the number of transistors on a CPU can be expected to double roughly every 18 months => now more cores
- It is a low-level language for multiprocessor development.
- Experience with C taught that a successful systems programming language ends up being used for application development.
- Go incorporates a number of high-level features, allowing developers to use it for things like web services or desktop applications, as well as very low-level systems.



# Who Uses Go?

- [Docker](#), a set of tools for deploying [Linux](#) containers
- [Ethereum](#), blockchain for the *Ether* cryptocurrency
- [InfluxDB](#), an open source database specifically to handle time series data with high availability and high performance requirements.
- [Juju](#), a service orchestration tool by [Canonical](#), packagers of [Ubuntu](#)
- [Kubernetes](#) container management system
- [OpenShift](#), a cloud computing platform as a service by [Red Hat](#)
- [Terraform](#), an open-source, multiple [cloud](#) infrastructure provisioning

# Who Uses Go?

- [Cloud Foundry](#), a platform as a service
- [Container Linux](#) (formerly CoreOS), a Linux-based operating system that uses [Docker](#) containers and [rkt](#) containers.
- [Couchbase](#), Query and Indexing services within the Couchbase Server
- [Dropbox](#), migrated some of their critical components from Python to Go
- [Heroku](#), for Doozer, a lock service
- [MongoDB](#), tools for administering MongoDB instances
- [Netflix](#), for two portions of their server architecture
- [Uber](#), for handling high volumes of geofence-based queries

# Why Go?

- *Minimalism* - Go language specification is only 50 pages, with examples, easy to read. Core language consists of a few **simple, orthogonal** features that can be combined in a relatively small number of ways.
- *Code transparency* - your need to understand your code:
  - you **always** need to know **exactly what** your coding is doing;
  - you **sometimes** need to **estimate the resources** (time and memory) it uses;
  - one standard code format, automatically generated by the **fmt** tool.
- *Compatibility* - Go 1 has succinct and strict compatibility guarantees for the core language and standard packages. BSD-style license.
- *Performance* - compiled language, single standalone binary, low latency garbage collection, optimized standard libraries, fast build, scales well.

# Go Main Features

- *Static typing* and *run-time efficiency* (like C++)
- *Syntax and environment patterns* more common in dynamic languages
- *Readability, usability* and *simplicity*
- *Fast compilation* times
- *High-performance networking* and *multiprocessing*
- Optional *concise variable declaration* and initialization through *type inference* (`x := 0` not `int x = 0`; or `var x = 0`).
- Remote *package management* (`go get`) and online *package documentation*.

# Distinctive Approaches to Particular Problems

- Go is *strongly* and *statically* typed with no implicit conversions, but the syntactic overhead is small by using simple type inference in assignments together with untyped numeric constants.
- An *interface* system in place of *virtual inheritance*, and *type embedding* instead of *non-virtual inheritance*.
- Structurally typed *interfaces* provide *runtime polymorphism* through *dynamic dispatch*.
- Programs are constructed from *packages* that offer clear code separation and allow efficient management of dependencies.
- Built-in concurrency primitives: light-weight processes (*goroutines*), *channels*, and the *select* statement

# Distinctive Approaches to Particular Problems

- A toolchain that, by default, produces statically linked *native binaries without external dependencies*.
- Built-in frameworks for *testing* and *profiling* are small and easy to learn, but still fully functional.
- It's possible to *debug* and *profile* an optimized binary running in production through an HTTP server.
- Go has *automatically generated documentation* with *testable examples*.

# Built-in Types

- Strings are provided by the language; a string behaves like a slice of bytes, but is immutable.
- Hash tables are provided by the language. They are called maps.



# Pointers and References

- Go offers pointers to values of all types, not just objects and arrays. For any type `T`, there is a corresponding pointer type `*T`, denoting pointers to values of type `T`.
- Arrays in Go are values. When an array is used as a function parameter, the function receives a copy of the array, not a pointer to it. However, in practice functions often use slices for parameters; slices are references to underlying arrays.
- Certain types (maps, slices, and channels) are passed by reference, not by value. That is, passing a map to a function does not copy the map; if the function changes the map, the change will be seen by the caller. In Java terms, one can think of this as being a reference to the map.

# Error Handling

- Instead of **exceptions**, Go uses errors to signify events such as end-of-file;
- And run-time **panics** for run-time errors such as attempting to index an array out of bounds.

# Object-Oriented Programming

- Go does not have classes with constructors. Instead of instance methods, a class inheritance hierarchy, and dynamic method lookup, Go provides **structs** and **interfaces**.
- Go allows **methods** on any type; no boxing is required. The **method receiver**, which corresponds to this in Java, can be a direct value or a pointer.
- Go provides two **access levels**, analogous to Java's public and package-private. Top-level declarations are **public** if their names start with an **upper-case letter**, otherwise they are **package-private**.

# Functional Programming. Concurrency

- Functions in Go are first class citizens. Function values can be used and passed around just like other values and function literals may refer to variables defined in a [enclosing function \(closure\)](#).
- Concurrency: Separate threads of execution, [goroutines](#), and [communication channels](#) between them, channels, are provided by the language.

# Omitted Features

- Go does not support implicit type conversion. Operations that mix different types require an explicit conversion. Instead Go offers Untyped numeric constants with no limits.
- Go does not support function overloading. Functions and methods in the same scope must have unique names. As alternatives, you can use optional parameters.
- Go has some built-in generic data types, such as slices and maps, and generic functions, such as append and copy. However, there is no mechanism for writing your own generic functions.



# Installing Go

Download, installation, environment setup



# Installation and Setup

- Download the Binary from: <https://golang.org/dl/>
- Windows - MSI installer. Installer should put the `c:\Go\bin` directory in your `PATH` environment variable.
- Create your workspace directory – e.g. `%USERPROFILE%\go`
- Setting environment variables under Windows – `GOPATH` (must not be the same path as your Go installation), `GOTMPDIR`, etc. Ex (Windows 10):
  - Open a command prompt (`Win + r` then type `cmd`) or a `powershell` window (`Win + i`).
  - Type: `setx GOPATH %USERPROFILE%\go`
  - OR (Go 1.13): `go env -w GOPATH=c:\go-work`
- Test your installation



# Hello World in Go

```
package main

import "fmt"

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

```
C:\> cd %USERPROFILE%\go\src\hello
```

```
C:\Users\Gopher\go\src\hello> go build
```

```
C:\Users\Gopher\go\src\hello> hello
```

```
Hello, world!
```

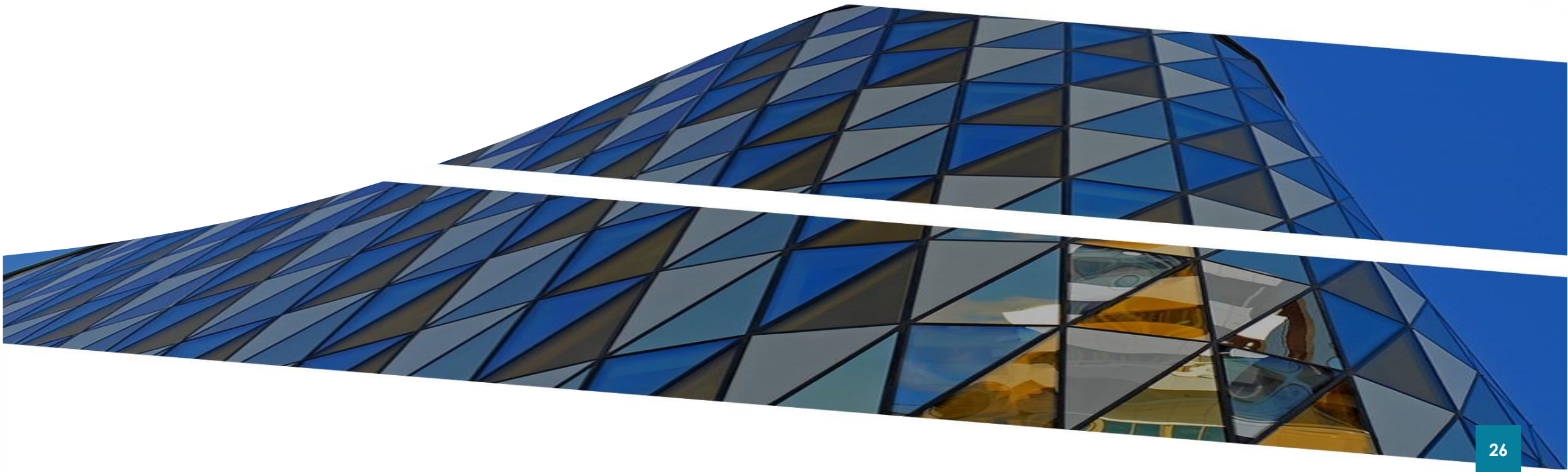
# Workspace Structure

```
bin/  
  hello          # command executable  
  outyet         # command executable  
src/  
  github.com/golang/example/  
    .git/         # Git repository metadata  
    hello/  
      hello.go    # command source  
    outyet/  
      main.go     # command source  
      main_test.go # test source  
    stringutil/  
      reverse.go  # package source  
      reverse_test.go # test source  
  golang.org/x/image/  
    .git/         # Git repository metadata  
    bmp/  
      reader.go   # package source  
      writer.go   # package source  
... (many more repositories and packages omitted) ...
```



# Go Basic Syntax

Download, installation, environment setup



# The Structure of a Go Source File

Go code is arranged in **packages**, which fill the roles of both libraries and header files in C

package main

Every program must contain a **main** package, which contains a **main()** function, which is the program entry point

import "fmt"

**fmt** package has been imported, any of its exported **types**, **variables**, **constants**, and **functions** can be used, prefixed by the package name; packages are imported when the code is linked, rather than when it is run; **access control** in Go is available only at package level.

func main() {

fmt.Println("Hello, world!")

}

**Println()** exported (public) function prints the text on the console

# Creating Simple Library Package

```
// Package stringutil contains utility functions for working with strings.  
package stringutil
```

```
// Reverse returns its argument string reversed rune-wise left to right.  
func Reverse(s string) string {  
    r := []rune(s)  
    for i, j := 0, len(r)-1; i < len(r)/2; i, j = i+1, j-1 {  
        r[i], r[j] = r[j], r[i]  
    }  
    return string(r)  
}
```

# Using It

```
package main

import "fmt"
import "github.com/iproduct/coursego/simple/stringutil"

func main() {
    s := "Hello Go World!"
    fmt.Println(s)
    fmt.Println(stringutil.Reverse(s))
}
```



# More Examples: Let's Write Some Code

- Variables
- Loops
- Functions
- Enums
- Structures and Methods
- Interfaces
- Polymorphism
- Casting
- Errors
- Http Client and Server

# Recommended Literature

- The Go Documentation - <https://golang.org/doc/>
- The Go Bible: Effective Go - [https://golang.org/doc/effective\\_go.html](https://golang.org/doc/effective_go.html)
- David Chisnall, *The Go Programming Language Phrasebook*, Addison Wesley, 2012
- Alan A. A. Donovan, Brian W. Kernighan, *The Go Programming Language*, Addison Wesley, 2016
- Nathan Youngman, Roger Peppé, *Get Programming with Go*, Manning, 2018
- Naren Yellavula, *Building RESTful Web Services with Go*, Packt, 2017

# Thank's for Your Attention!



Trayan Iliev

IPT – Intellectual Products & Technologies

<http://iproduct.org/>

<http://robolearn.org/>

<https://github.com/iproduct>

<https://twitter.com/trayaniliev>

<https://www.facebook.com/IPT.EACAD>