Introduction to Golang
Part 1

DOMINEUM

BLOCKCHAIN

Calistus Igwilo
https://linkedin.com/in/calistus-igwilo
https://twitter.com/CalistusIgwilo

Announcements

- MPlease join the 5thwork portal if you haven't
  - https://5thwork.com/courses/course-v1:NITDA+NITDA0001+2022Q3/about
  -
- Join the course Telegram group for announcements
  - https://t.me/+HzLMWqDbYcBhNTU8

- Precourse extended to 31st December, 2022

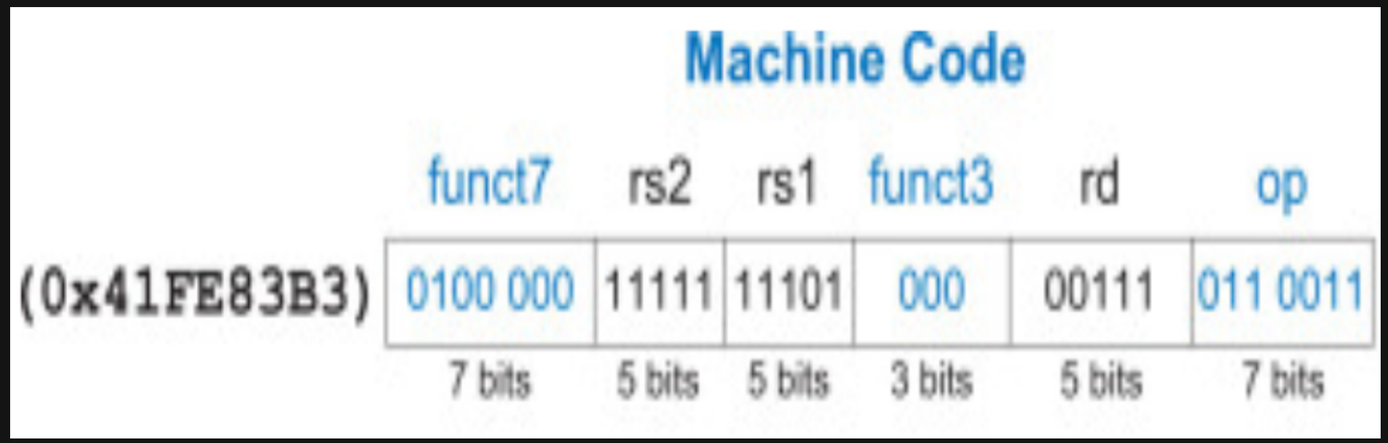
Why Learn Go?

- Code runs fast
- Garbage collection
- Simpler objects
- Conccurency

Software Translations

Machine Language

- CPU instructions represented in binary
  - 0100 0001 1111 1110 1000 0011 1011 0011 (0x41FE83B3)
- The seven least significant bits defines how to interprete the instruction (opcode)



Software Translations

Assembly Language

- CPU instructions are represented in mnemonics
  - instructions, varibles, addresses have names. Example: ADD, SUB, SUM
  - Eaiser to read
  - Equivalent to machine language

Software Translations

Assembly Language

Assembly Language Mnemonic	Machine Language Operation Code	Function Performed
A	5A	ADD
C	59	COMPARE
CVB	4F	CONVERT TO BINARY
CVD	4E	CONVERT TO DECIMAL
D	5D	DIVIDE
ED	DE	EDIT
L	58	LOAD
MVC	D2	MOVE CHARACTERS
M	5C	MULTIPLY
ST	50	STORE
S	5B	SUBTRACT
SVC	0A	SUPERVISOR CALL
TR	DC	TRANSLATE
ZAP	F8	ZERO AND ADD

Harris et al 2022, Digital design and computer architecture

Software Translations

High Level Languge

- Much easier to read
- Closer to human language
- Structured with syntax and semantics to describe the computing algorithm
- Examples: C, C++, Python, JavaScript, Go

Compiled Vs Translated

Compilation

- Translate instructions once before running the code
- Example: C, C++
- Saves time, as translation occurs only once

Compiled Vs Translated

Interpretation

- Translate instructions while code is executed
- Example: Python, JavaScript
- Requires an interpreter

Compiled Vs Translated

Demo

- Python for interpreter
- C for Compiler

Efficiency vs Ease of Use

- Compiled code is fast
- Interpreters make coding easier
  - Manage memory automatically
  - Infer variables
- Go is a good compromise

Garbage Collection

- Automatic memory management
  - Where should memory be allocated?
  - When can memory be dealocated?

Garbage Collection

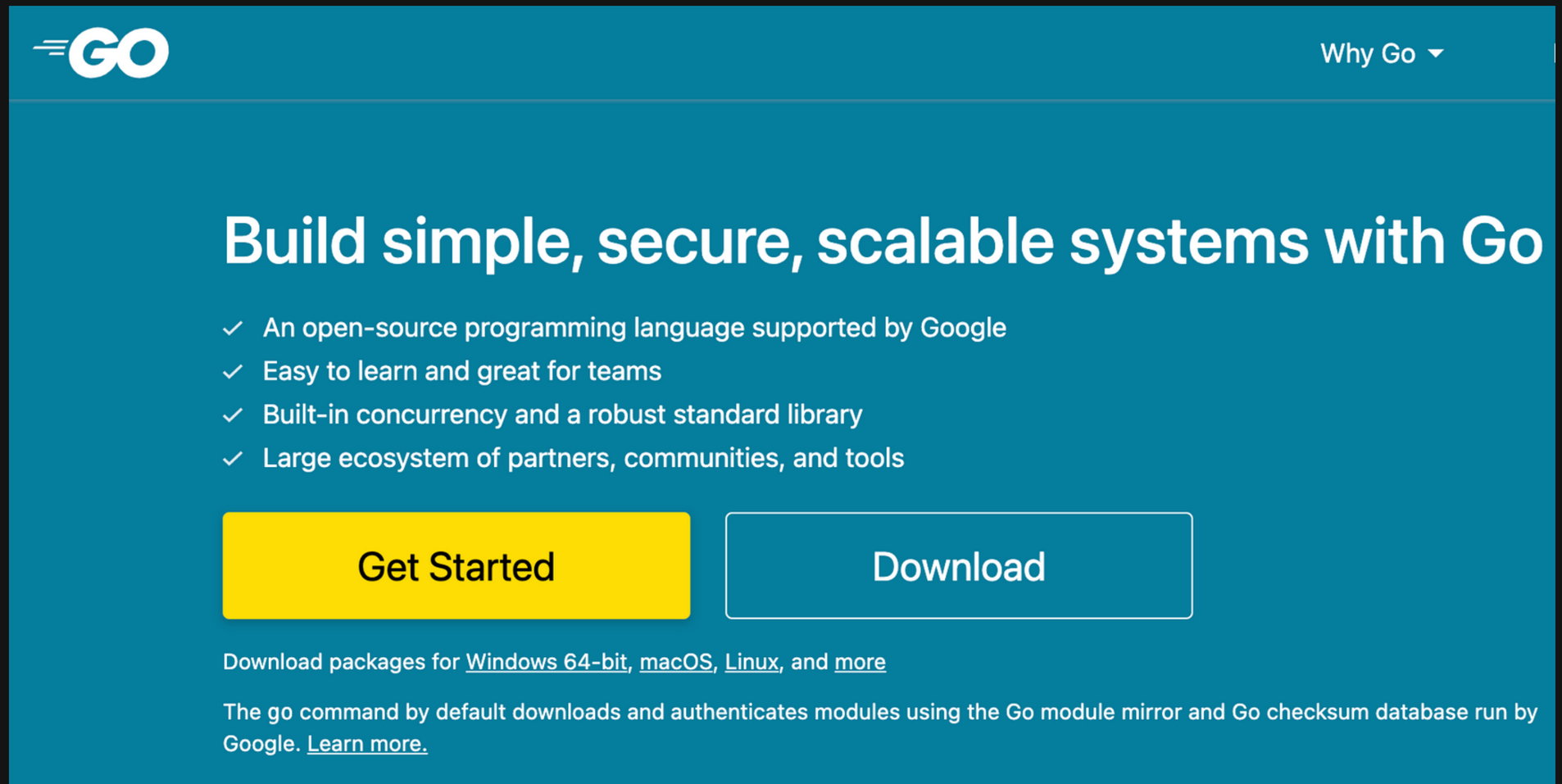
- Memory memory management is hard
  - Deallocate too early, false memory accesses
  - Deallocate too late, wasted memory

Garbage Collection

- Go includes garbage collection
  - Typicall done by interpreters

Installing Go

- https://go.dev



Installing Go

Downloads

After downloading a binary release suitable for your system, please follow the installation instructions.

If you are building from source, follow the source installation instructions.

See the release history for more information about Go releases.

As of Go 1.13, the go command by default downloads and authenticates modules using the Go module mirror and Go checksum database run by Google. See https://proxy.golang.org/privacy for privacy information about these services and the go command documentation for configuration details including how to disable the use of these servers or use different ones.

Featured downloads

Microsoft Windows
Windows 7 or later, Intel 64-bit processor
go1.19.4.windows-amd64.msi
(135MB)

Apple macOS (ARM64)
macOS 11 or later, Apple 64-bit processor
go1.19.4.darwin-arm64.pkg
(139MB)

Apple macOS (x86-64)
macOS 10.13 or later, Intel 64-bit processor
go1.19.4.darwin-amd64.pkg
(145MB)

Linux
Linux 2.6.32 or later, Intel 64-bit processor
go1.19.4.linux-amd64.tar.gz
(142MB)

Source
go1.19.4.src.tar.gz (25MB)

Workspaces

Hierarchy of directories

Common organisation is good for sharing

Workspaces

Three sub-directories

- Src - contains source files
- pkg - contains packages (libraries)
- bin - contains executables

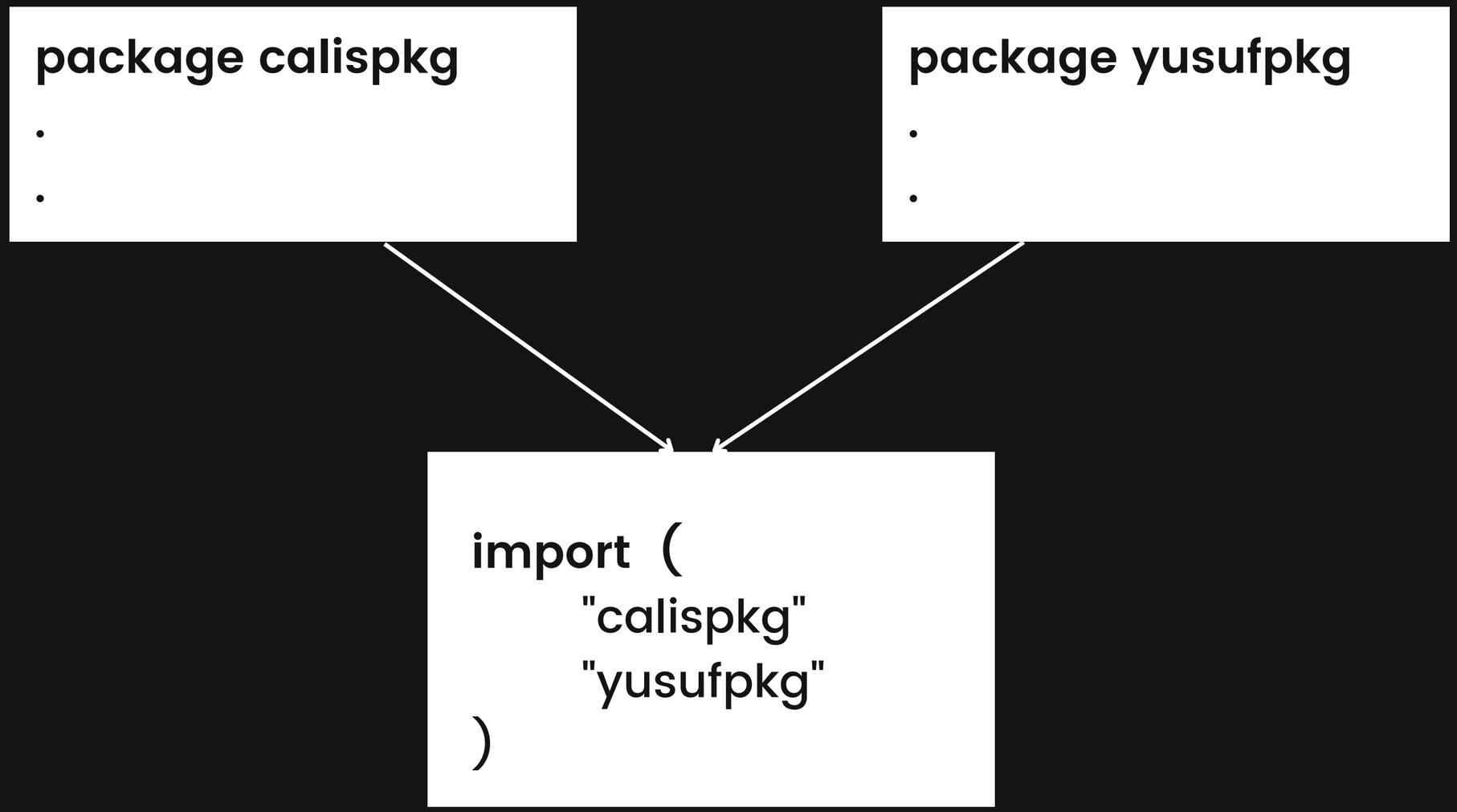
Recommended, not enforced

Packages

- Group of related source files
- Each package can be imported by other packages
- enables software reuse

Packages

First line of code identifies the package



Package main

- There must be one package called main
- Building the main package generates an executable
- main package needs a main() function
- main() is where exection starts

Package main

```
package main

import "fmt"
func main() {
    fmt.Printf("Hello, world\n")
}
```

Go Tool

import

- import keyword is used to access other packages
- Go standard library includes many packages
  - fmt
  -
- Searches directories specified by GOROOT and GOPATH

Go Tool

go build

- Compiles the program
  - arguements can be a list of packages of go files
  - creates an executable for the main package
  - .exe suffix for windows executables

Go Tool

go doc

- Prints documentation for the package
  -

go fmt

- formats go source files

go get

- downloads packages and installs them

Variables

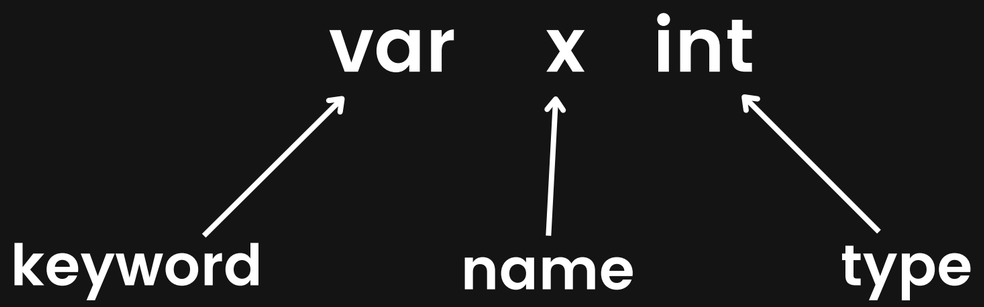
- container that holds data in the memory

Naming

- Names must start with a letter
- Any number of letters, digits, underscores
- Case sensitive
- Don't use kewords

Variables

- Must have a name and a type
- Must have declaration



- Can declare many on the same line

```
var x, y, int
```

Variables Types

- Types defines the values a variable might take and the operations that can be performed on it.
- Integer
  - Only integral values
  - Integer arithmetic (+, -, *, ...)
- floating point
  - Fractional decimal values
  - floating point arithmetic (+, -, *, ...)

Variables Types

- Strings
  - Bytes, character, sequences
  - String comparison, search, concat, etc

- Boolean
  - true or false

Variables Initialization

- Initialize in the declaration
  - var int = 0;

- Initialize after declaration
  - var x int
  - x = 0
- Unitialized variables gets 0 for its type
  - var x int // x = 0
  - var x string // x = ""