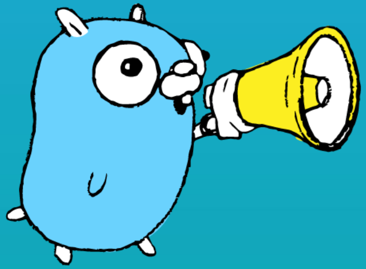
Fundamentals of Programming – Term 1/2020

Fundamentals of Programming

Asst.Prof.Orathai Sangpetch, Dr.Akkarit Sangpetch

KMITL / CMKL University

Computer Innovation Engineering



Today's (glorious) blather.

Administration	01
Grading & Course Policies	02
Class Project	03
Course Outcomes	04
About Programming	05
Go Tutorial	06

SECTION ONE

Course Administration

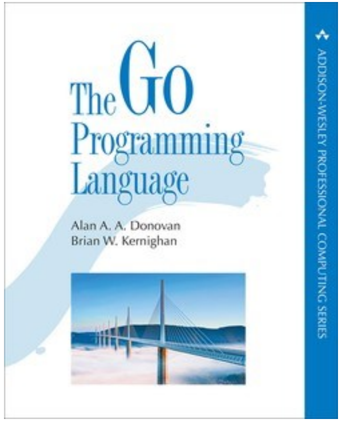
Administration & Personnel

- Class hours
  - Lecture:
    - Monday 8:45 am – 10:15 am
    - Wednesday 1 pm – 2:30 pm
  - Labs:
    - Tuesday 1 pm – 2:30 pm
    - Wednesday 2:30 pm – 4 pm
- Instructors
  - Dr. Orathai Sangpetch (orathai.sa@kmitl.ac.th)
  - Dr. Akkarit Sangpetch (akkarit.sa@kmitl.ac.th)
- TAs
  - Mr. Parnmet Daengphruan (parnmet@cmkl.ac.th)
  - Mr. Wachirawich Siripaktanakon (wachirawich@cmkl.ac.th)

Course Materials

- **Canvas LMS**: class materials (Lectures, Grades, HW Submissions)
  - https://kmitl.instructure.com/enroll/GMG8CP

- Textbook
  - Alan A. A. Donovan & Brian W. Kernighan;
    The Go Programming Language (2015)
    http://www.gopl.io/


The Go Programming Language
Alan A. A. Donovan
Brian W. Kernighan
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

- Supplements
  - Go website: https://golang.org/
  - Effective Go: https://golang.org/doc/effective_go.html

Course Grading

- 40% - Exams (Midterm / Final)
- 40% - Class project (2-4 people)
- 20% - Lab
- Scoring
  - ≥ 90% → at least A
  - ≥ 85% → at least B+
  - ≥ 80% → at least B
  - ≥ 75% → at least C+
  - ≥ 70% → at least C
  - ≥ 65% → at least D+
  - ≥ 60% → at least D

Academic Integrity

+ If you cheat on an exam, labs, assignments:
  - 1st time offender will get a zero grade for the assignment (both the author and the copier of duplicate works)
  - Repeated offender will get an **F** for the class + **Expulsion**

+ For lab, assignment and project submissions:
  - Copying your friends' code partially or fully will result in an **F**
  - Copying code from the Internet without proper attribution will result in an **F**
  - Anything else that could be considered plagiarism will get you zero or **F**

+ We encourage collaboration and discussion among you and your peers
  - Discuss and exchange ideas, but you must implement/code your work by yourself
  - Do not lookup lab/assignment answer on the Internet

Lab Submission Policy

+ If you submit your work late without proper cause, you will **get zero** on that assignment.
  - Sick leaves needs to be accompanied by an official document from your physician or appropriate authority.
  - Other proper causes (university events / other extra-curricular activities) can be discussed with the instructors and granted the permission ahead of time.

Class Project

- One class project (due before the end of the semester)
- Group of 3-4 people
- Use your acquired programming skills to solve a real-world problem
    - Must interact with one of the given gadgets
- Deadline
    - Project idea presentation & team members on August 19 @ 1pm
    - Project proposal submission and presentation on September 9 @ 1pm
- Project grading criteria
    - Individual scores (50%)
    - Group scores (50%)

Course Outcomes

1. To give students the tools and basic skills to solve a computational problem through the process of design, implementation, documentation and testing

   + Implement computer programs to perform basic computing tasks.
   + Understand and explain basic constructs of computer programming including variables, flow controls, functions
   + Perform basic debugging and identify flaws in computer programs
   + Implement computer programs which perform external I/O including files and networks

2. To give students an understanding of the breadth of computer science and how it exists or can be applied in the real world

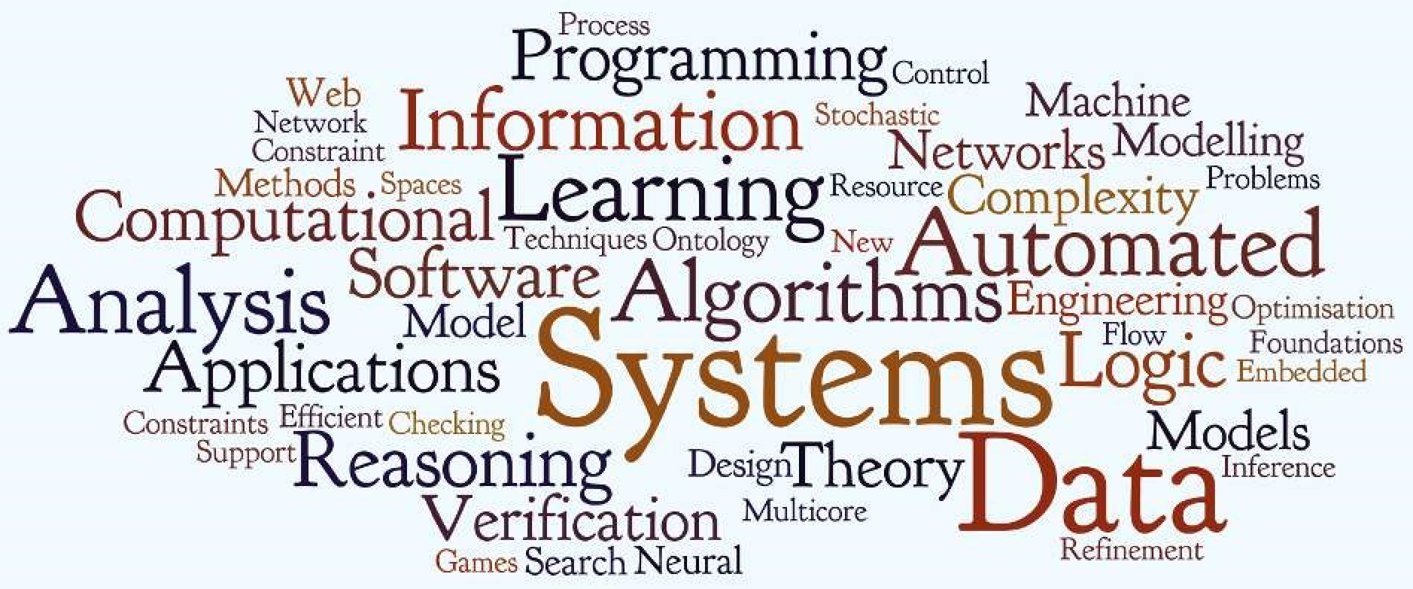
   + Identify applications of computer science and engineering
   + Explain the basics concepts for different programming styles including object-oriented programming, functional programming, imperative programming
   + Identify and utilize libraries or tools required to implement modern applications such as web or mobile applications
   + Model real-world problems to be solved using computer programs

SECTION TWO

Programming

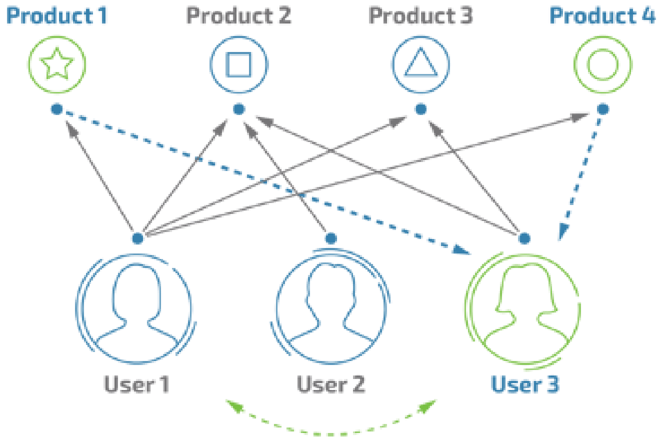
Programming

+ What is Programming?

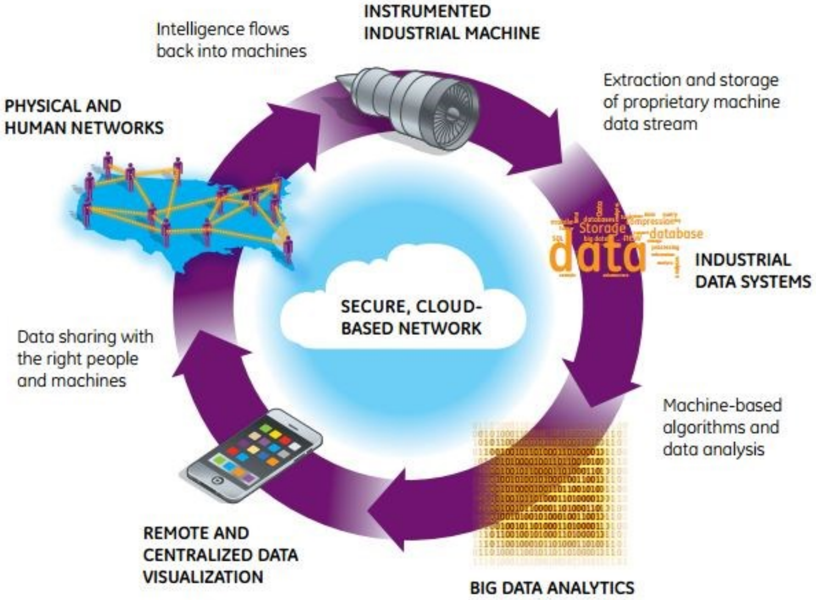


Information Retrieval and Social Network

Online Shopping & Recommendation



Internet of Things / Cyber-physical Systems



INSTRUMENTED INDUSTRIAL MACHINE

Extraction and storage of proprietary machine data stream

INDUSTRIAL DATA SYSTEMS

Machine-based algorithms and data analysis

BIG DATA ANALYTICS

REMOTE AND CENTRALIZED DATA VISUALIZATION

Data sharing with the right people and machines

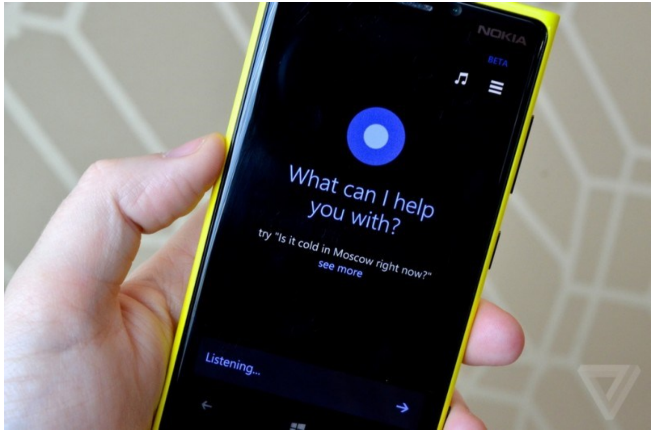
PHYSICAL AND HUMAN NETWORKS

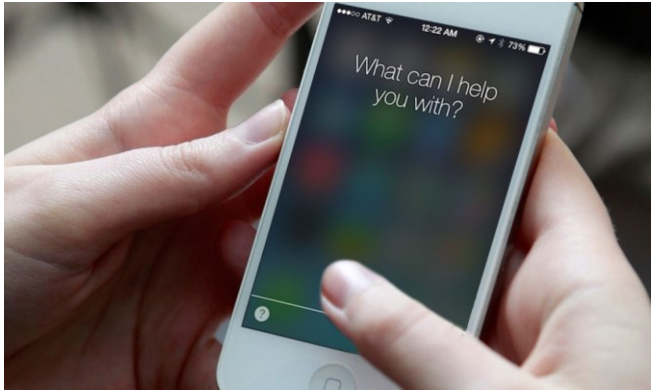
Intelligence flows back into machines

SECURE, CLOUD-BASED NETWORK

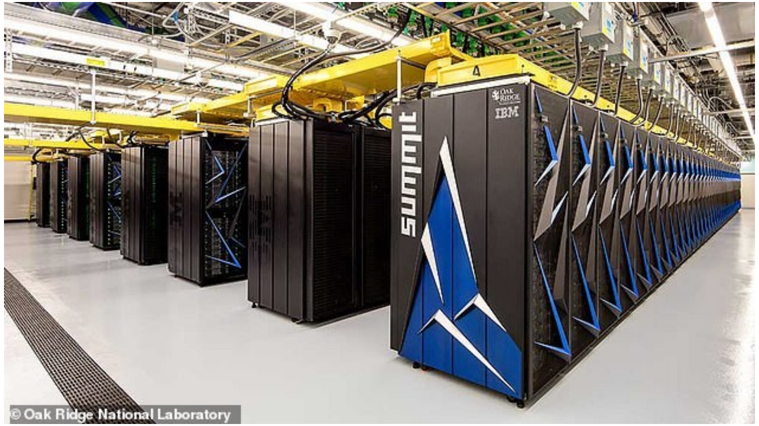
Games & Entertainment

Personal Assistant / Autonomous Vehicles


NOKIA
BETA
What can I help you with?
try "Is it cold in Moscow right now?"
see more
Listening...



Cloud Computing / HPC / Infrastructure



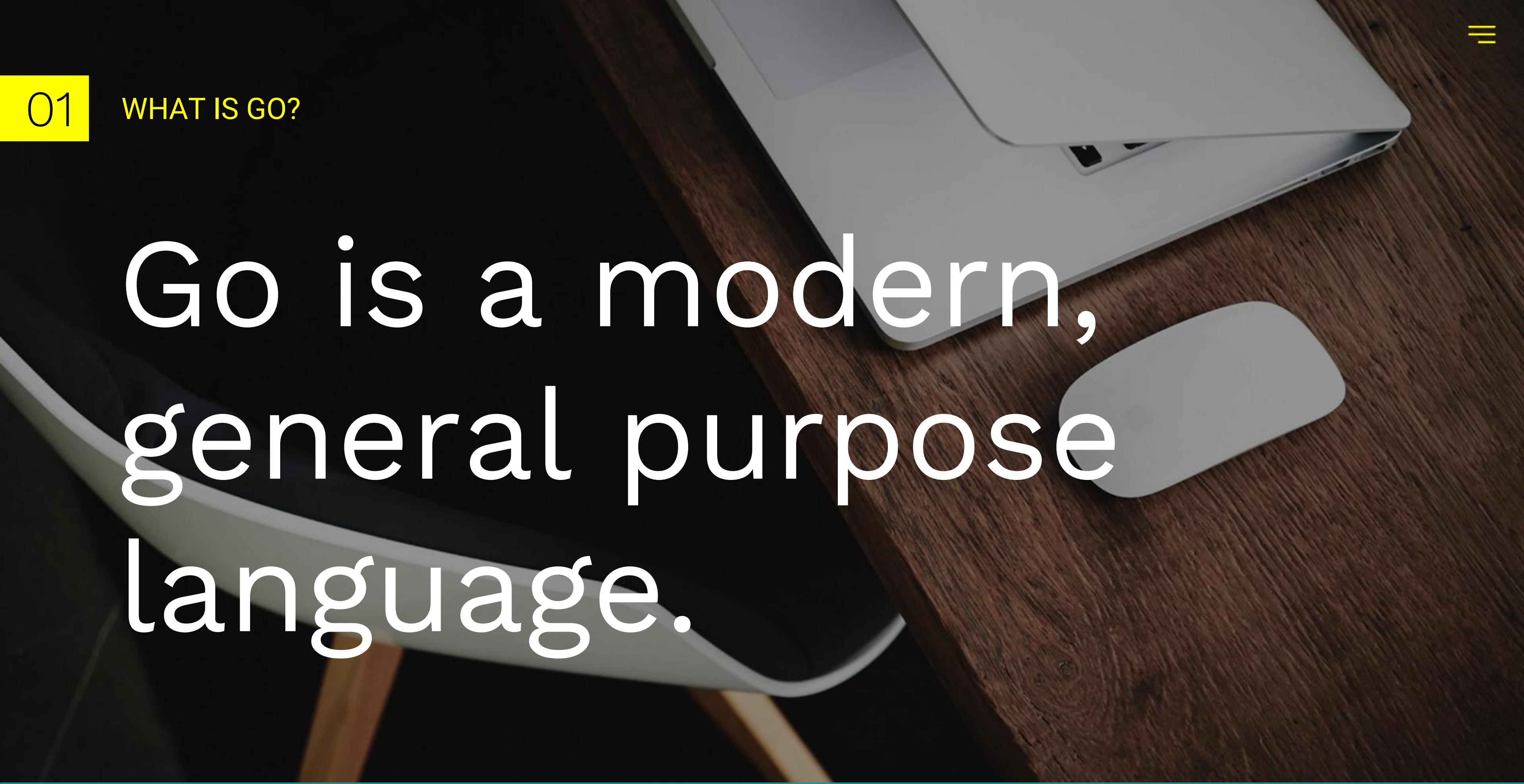
© Oak Ridge National Laboratory

SECTION THREE

Go Tutorial

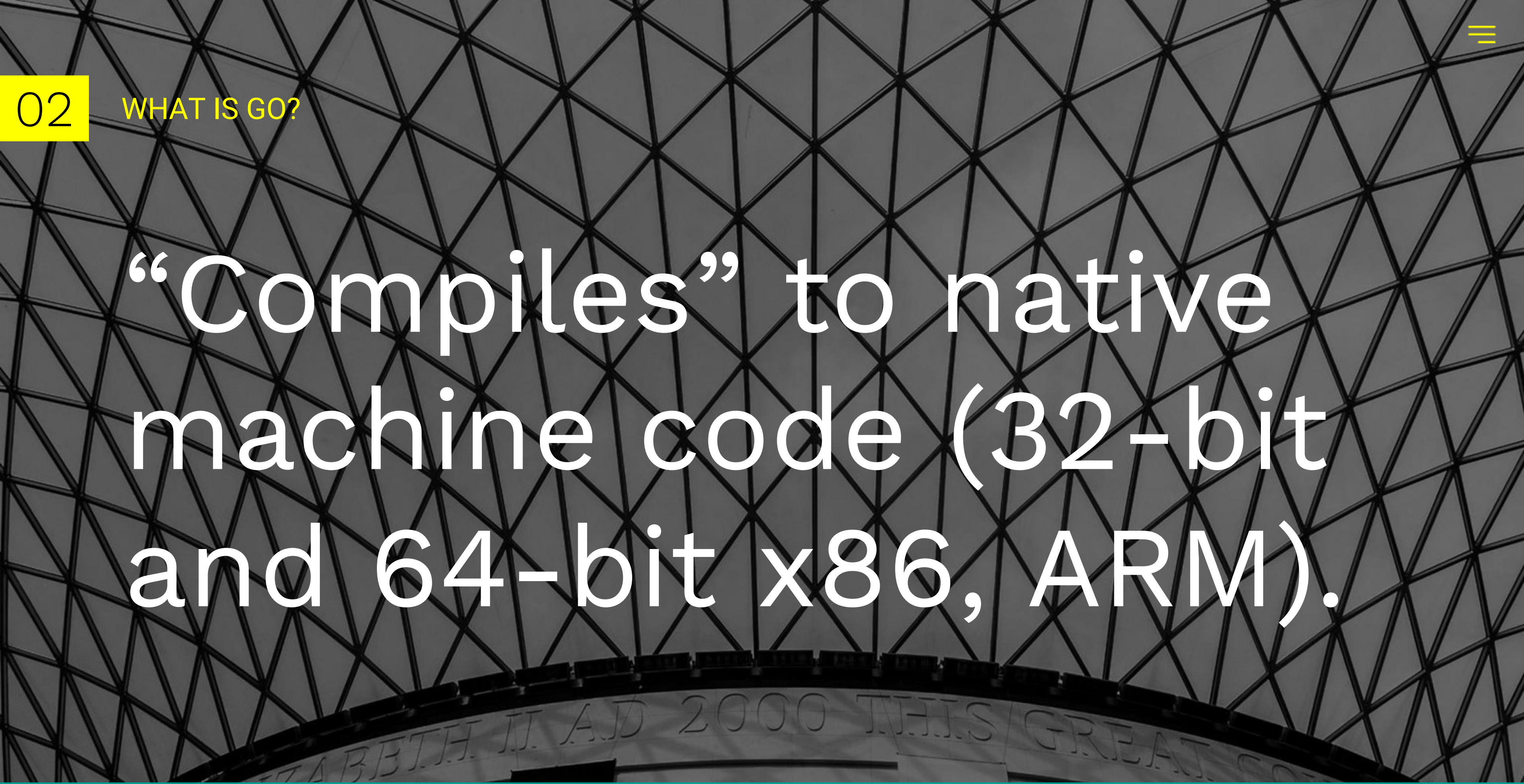
Why Go?

- Developer productivity of a dynamic language with the speed, safety, and reliability of a static language
- Easy to learn & readable
- Has a vibrant, welcoming community, spanning open-source developers, startups, large companies, and universities
- The language of the cloud
- KMITL GO ☺



01 WHAT IS GO?

Go is a modern, general purpose language.

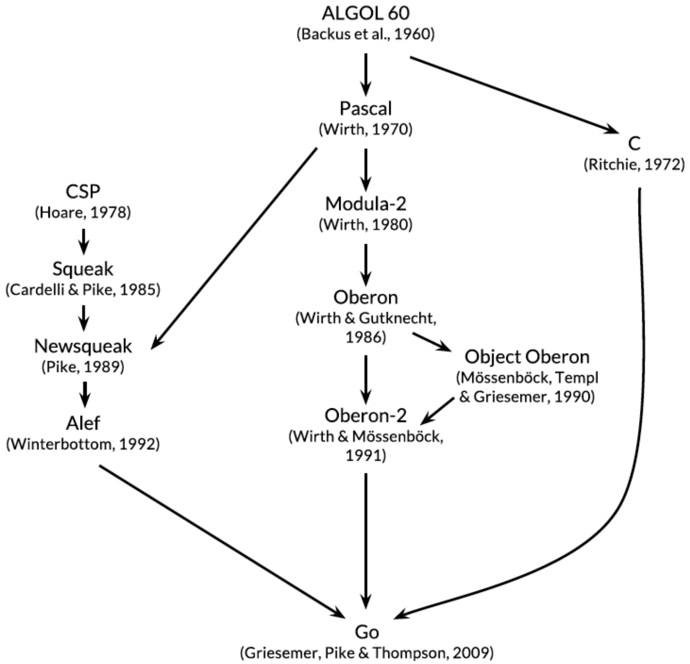


“Compiles” to native machine code (32-bit and 64-bit x86, ARM).

Lightweight syntax.

Go’s Values & Origins

- Thoughtful
- Simple
- Efficient
- Reliable
- Productive
- Friendly



Go = “ C for the 21st century ”

TENETS OF GO DESIGN

Simplicity.

Each language feature should be easy to understand.

TENETS OF GO DESIGN

Orthogonality.

Go's features should interact in predictable and consistent ways.

Getting Started

- Integrated Development Environment - Visual Studio Code: https://code.visualstudio.com/
- Go compiler & tool installation: https://golang.org/doc/install
- Take an on-line tour: https://tour.golang.org/

Project Layout & Version Control

- **For projects: use modules, project layout & version control (git)**

go.mod	Module declaration
go.sum	Module hashes
.gitignore	git Ignore
/cmd	Main app/executable
/internal	Private app & library
/pkg	Sharable libraries
/build	Packaging & CI
/test	Additional tests
/vendor	Dependencies

Go standard community project layout
https://github.com/golang-standards/project-layout

Go Modules
https://blog.golang.org/using-go-modules

Git version control
https://git-scm.com/

GitHub (for hosting code repository)
https://github.com/

Learn Git: Git-It
https://github.com/jlord/git-it-electron

Hello World

Test your installation

```
hello.go
  package main
  import “fmt”

  func main() {
    fmt.Println(“Hello, โก”)
  }

$ go run hello.go
$ go build hello.go
$ ./hello
```

Go tools

run compile & run

build compile

fmt format source

get load modules

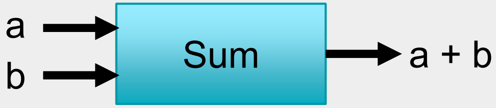
Ready to jump in ? ...

Function

```
package main
import “fmt”

func Sum(a, b int64) int64 {
    return a + b
}

func main() {
     fmt.Println(Sum(10, 20))
}
```



Function wraps a sequence of statements as a unit that can be called elsewhere in a program.

Package *main* = executable program

Function *main* = beginning of program execution

Command-Line Arguments: Loop, Slices, Assignments

“echo” prints its command line arguments on a single line

```
// Echo print its command-line arguments.
package main

import (
  “fmt”
  “os”
)

func main() {
  var s, sep string
  for i:=1; i < len(os.Args); i++ {
    s += sep + os.Args[i]
    sep = “ ”
  }
  fmt.Println(s)
}
```

var declares variables (with zero values)

:= short variable declaration

for loop statement in Go
for init; condition; post {
// statements
}
for condition {
// statements
}
= variable assignment
+= assignment operator

os.Args is a string **slice** ([]string) containing a list of argument strings of length *len(os.Args[i])*, accessible via index *os.Args[i]*

Finding Duplicate Lines

if, map and reading from keyboard

```
package main

import (
  “bufio”
  “fmt”
  “os”
)

func main() {
  counts := make(map[string]int)
  input := bufio.NewScanner(os.Stdin)
  for input.Scan() {
    counts[input.Text()]++
  }
  for line, n := range counts {
    if n > 1 {
      fmt.Printf(“%d\t%s\n”, n, line)
    }
  }
}
```

for can iterate over a range of values -> yield index + value for each iteration

map holds a set of key/value pairs

if statement executes the code if the condition is met

```
if condition {
  // statements if condition holds
} else {
 // other statements to execute
}
```

ExportFunction from packages starts with capital letter (bufio.NewScanner)

TYPES & METHODS (OOP)

You can define named type & methods on any type:

```
type Abser interface {
    Abs() float64
}

type MyFloat float64

func (m MyFloat) Abs() float64 {
    f := float64(m)
    if f < 0 {
        return -f
    }
    return f
}

func PrintAbs(a Abser) {
    fmt.Printf(“Absolute value: %.2f\n”, a.Abs())
}

f := MyFloat(-42)
f.Abs() // == 42.0
printAbs(f)
```

(m MyFloat) is called the ‘receiver’ for method Abs()

An interface type defines a set of methods (behaviors)

Types that implement those methods implicitly implements the interface

Fetching URLs Concurrently

Goroutines & concurrent programming

```
package main

import (
  ...
)

func main() {
  ch := make(chan string)
  for _, url := range os.Args[1:] {
    go fetch(url, ch)
  }
  for range os.Args[1:] {
    fmt.Println(<-ch)           // read from channel ch
  }
}

func fetch(url string, ch chan<- string) {
  resp, err := http.Get(url)
  if err != nil {
    ch <- fmt.Sprint(err)       // write to channel ch
    return
  }
  nbytes, err := io.Copy(ioutil.Discard, resp.Body)  // read but discard body content
  resp.Body.Close()  // close the response stream to avoid memory leak
  if err != nil {
    ch <- fmt.Sprintf(“while reading %s: %v”, url, err)
    return
  }
  ch <- fmt.Sprintf(“%7d %s”, nbytes, url)
}
```

go is used to invoke a concurrent function execution, aka. goroutine

chan is a channel used to send/receive values among goroutines.

A goroutine send or receive execution blocks until another goroutine receives or sends.

ioutil.ReadAll or *io.Copy* read or copy content to writer from reader.

A Web Server

Go’s standard libraries make it easy to write a web server

```
package main

import (
  “fmt”
  “log”
  “net/http”
)

func main() {
  http.HandleFunc(“/”, handler)
  log.Fatal(http.ListenAndServe(“localhost:8000”, nil))
}

func handler(w http.ResponseWriter, r *http.Request) {
  fmt.Fprintf(w, “URL.Path = %q\n”, r.URL.Path)
}
```

A variable is a piece of storage containing a value
x := 1

A pointer value is the address of a variable
p := &x
**p = 2* is the same as *x = 2*

http.Request* is a **pointer to http.Request type – passing pointers to function avoid copying of large structures

handler Go functions can be passed as parameters, just like variables

> Go's purpose is to make its designers' programming lives better.

ROB PIKE