Let's get Go-ing

An introduction to Go programming for COS 316
Today's Agenda

Just enough Go to get started on Assignment 1.

- What is Go?
- Variables, loops, and functions in Go
- Navigating the standard library documentation
Why learn Go?
Why learn Go?

Go is a programming language designed for large, distributed systems.
Why learn Go?

Go is a programming language designed for large, distributed systems. Widely used in industry.
Why learn Go?

Go is a programming language designed for large, distributed systems.

Widely used in industry.

Features native, efficient concurrency primitives (i.e., goroutines and channels).
Okay, let's write our first program
Variables

https://go.dev/play
package main

func main() {

}
package main

func main() {
    var a int = 3
}

VARIABLES
package main

func main() {
    var a int = 3
}

Variables

Variable types come after variable names
package main

func main() {
    var a int = 3
    var b = 2
}

Variable types come after variable names
package main

func main() {
    var a int = 3
    var b = 2
}

Variable types come after variable names

Variable types can be omitted and inferred
package main

func main() {
    var a int = 3
    var b = 2
    c := 1
}

Variable types come \textit{after} variable names

Variable types can be omitted and inferred
package main

func main() {
    var a int = 3
    var b = 2
    c := 1
}

Variable types come \textit{after} variable names

Variable types can be omitted and inferred

A shorthand for `var c =` is `c :=`
package main

func main() {
    var a int = 3
    var b = 2
    c := 1
    var d int
}

Variables

Variable types come after variable names

Variable types can be omitted and inferred

A shorthand for 'var c =' is 'c :='
package main

func main() {
    var a int = 3
    var b = 2
    c := 1
    var d int
}

variables

Variable types come after variable names

Variable types can be omitted and inferred

A shorthand for 'var c =' is 'c :='

Can choose to accept default value (i.e., 0)
Variables

```go
class main

func main() {
    var a int = 3
    var b = 2
    c := 1
    var d int
    var e, f int = -1, -2
}
```

Variable types come after variable names

Variable types can be omitted and inferred

A shorthand for 'var c = ' is 'c :='

Can choose to accept default value (i.e., 0)
package main

func main() {
    var a int = 3
    var b = 2
    c := 1
    var d int
    var e, f int = -1, -2
}

**Variables**

Variable types come after variable names

Variable types can be omitted and inferred

A shorthand for 'var c =' is 'c :='

Can choose to accept default value (i.e., 0)

Can declare and init. multiple vars in 1 line
package main

func main() {
    var a int = 3
    var b = 2
    c := 1
    var d int
    var e, f int = -1, -2
}

Okay, looks good!
Let's run our code.

Variable types come after variable names
Variable types can be omitted and inferred
A shorthand for 'var c = ' is 'c := '
Can declare and init. multiple vars in 1 line
Can accept default value (i.e., 0)
package main

func main() {
    var a int = 3
    var b = 2
    c := 1
    var d int
    var e, f int = -1, -2
}

Okay, looks good!
Let's run our code.

> go run main.go

Variable types come after variable names.

Variable types can be omitted and inferred.

Can declare and init. multiple vars in 1 line.
package main

func main() {
    var a int = 3
    var b = 2
    c := 1
    var d int
    var e, f int = -1, -2
}

Variables

Variable types come after variable names

Variable types can be omitted and inferred

Shorthand for var x = is x :=

Can choose to accept default value (i.e., 0)

Compiler says nope!

./main.go:4:7: a declared and not used
./main.go:5:7: b declared and not used
./main.go:6:3: c declared and not used
./main.go:7:7: d declared and not used
./main.go:8:7: e declared and not used
./main.go:8:10: f declared and not used

default value (i.e., 0)

Can declare and init. multiple vars in 1 line
package main

func main() {
    var a int = 3
    var b = 2
    c := 1
    var d int
    var e, f int = -1, -2
}

Go prevents you from compiling code with unused variables, so let's print them out

Can declare and init. multiple vars in 1 line

Variable types come after variable names

Variable types can be omitted and inferred

A shorthand for 'var c = ' is 'c := ' Can choose to accept default value (i.e., 0)
package main

func main() {
    var a int = 3
    var b = 2
    c := 1
    var d int
    var e, f int = -1, -2
}

Variables

Variable types come after variable names

Variable types can be omitted and inferred

A shorthand for 'var c =' is 'c :='

Can choose to accept default value (i.e., 0)

Can declare and init. multiple vars in 1 line
package main

import "fmt"

func main() {
    var a int = 3
    var b = 2
    c := 1
    var d int
    var e, f int = -1, -2
}

Variable types come after variable names

Variable types can be omitted and inferred

A shorthand for 'var c =' is 'c :='

Can choose to accept default value (i.e., 0)

Can declare and init. multiple vars in 1 line
package main

import "fmt"

func main() {
    var a int = 3
    var b = 2
    c := 1
    var d int
    var e, f int = -1, -2

    fmt.Println(a, b, c)
}

Variables

Variable types come after variable names

Variable types can be omitted and inferred

A shorthand for 'var c =' is 'c :='

Can choose to accept default value (i.e., 0)

Can declare and init. multiple vars in 1 line
package main
import "fmt"

func main() {
    var a int = 3
    var b = 2
    c := 1
    var d int
    var e, f int = -1, -2

    fmt.Println(a, b, c)
    fmt.Println(d, e, f)
}

**Variables**

Variable types come after variable names

Variable types can be omitted and inferred

A shorthand for 'var c =' is 'c :='

Can choose to accept default value (i.e., 0)

Can declare and init. multiple vars in 1 line
Let's see this in action!
"Go" to go.dev/play and try out some variable declarations.
Play time!

"Go" to go.dev/play and try out some variable declarations.

Here are some ideas.
1. Can you declare multiple variables with different types on the same line?

Play time!

"Go" to go.dev/play and try out some variable declarations.

Here are some ideas.
1. Can you declare multiple variables with different types on the same line?

2. Can you infer the types of variables when declaring more than one on a line?

Play time!

"Go" to go.dev/play and try out some variable declarations.

Here are some ideas.
1. Can you declare multiple variables with different types on the same line?

2. Can you infer the types of variables when declaring more than one on a line?

3. What does fmt.Println() print when it's given multiple arguments?

Play time!

"Go" to go.dev/play and try out some variable declarations. Here are some ideas.
package main

func main() {
}

Loops
package main

import "fmt"

func main() {
    for i := 1; i <= 3; i++ {
        fmt.Println(i)
    }
}
package main

import "fmt"

func main() {
    for i := 1; i <= 3; i++ {
        fmt.Println(i)
    }
}

'for' loops work like in Java/C, but don't require ()

Must use {}, even for 1-line loops
package main

import "fmt"

func main() {
    for i := 1; i <= 3; i++ {
        fmt.Println(i)
    }
    i := 4
    for i <= 10 {
        fmt.Println(i)
        i++
    }
}
package main

import "fmt"

func main() {
    for i := 1; i <= 3; i++ {
        fmt.Println(i)
    }
    i := 4
    for i <= 10 {
        fmt.Println(i)
        i++
    }
}
package main

import "fmt"

func main() {
    for i := 1; i <= 3; i++ {
        fmt.Println(i)
    }
    i := 4
    for i <= 10 {
        fmt.Println(i)
        i++
    }
    for {
        fmt.Println("done!")
        break
    }
}
package main

import "fmt"

func main() {
    for i := 1; i <= 3; i++ {
        fmt.Println(i)
    }
    i := 4
    for i <= 10 {
        fmt.Println(i)
        i++
    }
    for {
        fmt.Println("done!")
        break
    }
}
Let's try it ourselves
Let's Get Loopy

Navigate to go.dev/play and write a few Go loops.
1. Does the scoping of the index variable in a Go 'for' loop extend beyond the loop?

Let's Get Loopy

Navigate to go.dev/play and write a few Go loops.
1. Does the scoping of the index variable in a Go 'for' loop extend beyond the loop?

2. Can you skip the conditional part in a 'for' loop but still use the init and post statements?

Let's Get Loopy

Navigate to go.dev/play and write a few Go loops.
1. Does the scoping of the index variable in a Go 'for' loop extend beyond the loop?

2. Can you skip the conditional part in a 'for' loop but still use the init and post statements?

3. Does Go support 'labeled breaks' that let you choose which loop to leave?

Let's Get Loopy

Navigate to go.dev/play and write a few Go loops.
Functions
func f(a int, b int) int {
    return a + b
}
A function's return type is listed after its args
func f(a int, b int) int {
    return a + b
}

func g(a, b int) int {
    return a * b
}

A function's return type is listed after its args.
A function's return type is listed after its args.

If args are same type, can specify type once at end.
A function's return type is listed after its args.

If args are same type, can specify type once at end.

```go
func f(a int, b int) int {
    return a + b
}

func g(a, b int) int {
    return a * b
}

func h(a, b int) (int, int) {
    return f(a, b), g(a, b)
}
```
func f(a int, b int) int {
    return a + b
}

func g(a, b int) int {
    return a * b
}

func h(a, b int) (int, int) {
    return f(a, b), g(a, b)
}

A function's return type is listed after its args

If args are same type, can specify type once at end

Functions can return more than one result
```go
func f(a int, b int) int {
    return a + b
}

func g(a, b int) int {
    return a * b
}

func h(a, b int) (int, int) {
    return f(a, b), g(a, b)
}

func main() {
    a, b := h(1, 2)
    _, c := h(3, 4)
}
```

A function's return type is listed after its args.

If args are same type, can specify type once at end.

Functions can return more than one result.
func f(a int, b int) int {
    return a + b
}

func g(a, b int) int {
    return a * b
}

func h(a, b int) (int, int) {
    return f(a, b), g(a, b)
}

func main() {
    a, b := h(1, 2)
    _, c := h(3, 4)
}
Last programming exercise!
1. Does Go allow you to use '_' to ignore all the return values of a function?

2. Can you use recursion with a function that returns multiple values?

3. Does Go require a return value for each function?

Go Functions

Let's get back to go.dev/play and write a few programs using functions in Go.
Go Standard Library
All Go programs have access to a massive standard library of packages. (See pkg.go.dev/std)
All Go programs have access to a massive standard library of packages. (See pkg.go.dev/std)

This collection of officially supported packages is one of the reasons Go is a useful language for systems programmers.
Reading The Documentation
Navigating the documentation is hard.
Navigating the documentation is hard.

There's a lot of it and you'll be learning about the language as you read it.
Navigating the documentation is hard.

There's a lot of it and you'll be learning about the language as you read it.

Expect to spend some time pouring over it.
External Sources
Googling is allowed, even encouraged, in this course. You may use any online resource.
Googling is allowed, even encouraged, in this course. You may use any online resource.

If you base a significant portion of your code on it, cite it in a comment in your code.
External Sources

Googling is allowed, even encouraged, in this course. You may use any online resource. If you base a significant portion of your code on it, cite it in a comment in your code. Search for “golang” instead.
Let's see the docs
1. Find some “interesting” packages

2. Can you experiment using the provided examples?

Doc Hunt

Navigate to pkg.go.dev

Use go.dev/play
Questions?

Please don't hesitate to ask!
Additional Resources

- go.dev
- go.dev/play
- gobyexample.com
- "Learn Go Programming" (7 hour YouTube tutorial)
ASSIGNMENT 0

Ungraded!

Set up common development environment

○ Go, Git, etc.

○ Necessary for precepts and assignments
GIT & GO

- Command line Git
- Desktop Git
- Git Tutorial
- Git Cheatsheet
- Download Go