Intro to programming

Tom Arrell

What to expect?

- We will assume no prior knowledge of programming initially, however, we will expect to build upon knowledge from previous lessons.

What to expect?

- We will assume no prior knowledge of programming initially, however, we will expect to build upon knowledge from previous lessons.
- We will cover theory only when it becomes relevant to achieve our goals

What to expect?

- We will assume no prior knowledge of programming initially, however, we will expect to build upon knowledge from previous lessons.
- We will cover theory only when it becomes relevant to achieve our goals
- We *will* be writing as much code as possible, learn by doing

What to expect?

- We will assume no prior knowledge of programming initially, however, we will expect to build upon knowledge from previous lessons.
- We will cover theory only when it becomes relevant to achieve our goals
- We *will* be writing as much code as possible, learn by doing
- Please, please ask questions if things are not clear, or go ahead if you feel like things are too slow

Prerequisites

There are a few things you need in order to follow along smoothly.

- A code editor

Prerequisites

There are a few things you need in order to follow along smoothly.

- A code editor
  - VSCode is recommended

Prerequisites

There are a few things you need in order to follow along smoothly.

- A code editor
  - VSCode is recommended
- The Go compiler, with your `$GOPATH` variable configured

What is programming?

Programming is fundamentally about giving instructions to a computer.

There are many different flavours of these instructions. A few common ones you may have heard about...

- Assembly
- C
- Golang, what we'll be learning
- Java
- Python
- Javascript

About those instructions

Computer are about manipulating memory

All programs are fundamentally just a list of instructions in binary (1 or 0) format, which the computer will understand.

e.g.

Add two numbers together

$$ADD = 00000000$$

Multiply two signed numbers together (i.e. includes negative)

$$MUL = 01101001$$

Writing binary is hard so we need a way to make it easier for humans

What is Golang?

Don't worry if these don't make sense just yet, they will soon.

- Golang is a 11 year old language, designed by Robert Griesemer, Rob Pike, and Ken Thompson at Google

What is Golang?

Don't worry if these don't make sense just yet, they will soon.

- Golang is a 11 year old language, designed by Robert Griesemer, Rob Pike, and Ken Thompson at Google
- It is compiled

What is Golang?

Don't worry if these don't make sense just yet, they will soon.

- Golang is a 11 year old language, designed by Robert Griesemer, Rob Pike, and Ken Thompson at Google
- It is compiled
- Garbage collected

What is Golang?

Don't worry if these don't make sense just yet, they will soon.

- Golang is a 11 year old language, designed by Robert Griesemer, Rob Pike, and Ken Thompson at Google
- It is compiled
- Garbage collected
- Statically typed

What is Golang?

Don't worry if these don't make sense just yet, they will soon.

- Golang is a 11 year old language, designed by Robert Griesemer, Rob Pike, and Ken Thompson at Google
- It is compiled
- Garbage collected
- Statically typed
- Syntactically very similar to C

Compilation

Compilation is the process of taking code that a human can read, and outputting code that a computer can read.

Compilation

Compilation is the process of taking code that a human can read, and outputting code that a computer can read.

This will leave you with what we call a *binary*. Simply, a file containing all of the instructions.

[1] There is a bit more subtlety to this, but the simplification is fine for now

Compilation

Compilation is the process of taking code that a human can read, and outputting code that a computer can read.

This will leave you with what we call a *binary*. Simply, a file containing all of the instructions.

This *binary* will then be executable by the target machine. Think *.exe* on Windows, and an *Application* on Mac.[1]

[1] There is a bit more subtlety to this, but the simplification is fine for now

Practical: Part 1

Now that you know the basics, let's compile your first program in Golang.

1. Create a new directory anywhere, name it `helloworld`
2. Open up your code editor (VSCode)
3. Within VSCode, open the project in the directory you just created
4. Bring up the terminal within the project, Mac: *CTRL* + ‘
5. In the terminal, type: `go mod init helloworld`

Done? Great, please help someone next to you.

Practical: Part 2

6. Create a new file called `main.go`
7. Type the following:

```go
package main

import "fmt"

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

8. Back in the terminal, type: `go run .`

If you see "Hello, World!" printed out in your console. Everything worked!

Practical: Explanation

Congrats on writing your first Go program!

Now let's dig into what's actually going on...

Variables

Variables are little containers which you can put information into.

[2] This is only applicable to *statically* typed languages

Variables

Variables are little containers which you can put information into.

This information could be anything, but you must tell the compiler what *type* you want it to be.[2]

[2] This is only applicable to *statically* typed languages

Types

The basic types in Golang are:

Name	Type	Description
Boolean	bool	A boolean can be either *true* or *false*
String	string	A string is an set of letters
Integer	int...	Many different types, but all are *whole* numbers
Byte	byte	Another name for a number made up of 8 bits
Float	float...	Multiple types, but all represent numbers which can have decimal points

There are a couple more... But out of scope of this introduction.

Strings

We've already used a *string* by now. Look back at:

```
fmt.Println("Hello, World!")
```

One final exercise

Outputting things from your program is fun, but let's have a quick go at taking some *input*.

Reading Input

Let's modify our program a little bit.

```go
package main

import (
    "bufio"
    "fmt"
    "os"
)

func main() {
    scanner := bufio.NewScanner(os.Stdin)

    fmt.Print("What's your name? ")
    scanner.Scan()

    fmt.Println("Hello,", scanner.Text())
}
```

Once you've done that, feel free to run the command `go run .` from your terminal again and see what happens!

You should now be asked for your name. Type it in and press `ENTER`.

Result

```
$ go run .
What's your name? Tom
Hello, Tom
```

lesson 2, fin

On to more advanced things next week

Questions?